Allen-Bradley

Logix5000 Controllers Design Considerations

1756-Lx, 1769-Lx, 1789-Lx, 1794-Lx, PowerFlex 700S

Reference Manual

Rockwell Automation

Common Characteristic:	1756 ControlLogix	1769 CompactLogix	1789 SoftLogix5800	1794 FlexLogix	PowerFlex 700S DriveLogix
controller tasks • continuous • periodic • event	• 32 tasks (only 1 continuous) • event tasks: supports all event triggers	• 1769-L35E: 8 tasks • 1769-L32E: 6 tasks • 1769-L31: 4 tasks • 1769-L20, -L30: 4 tasks • only 1 task can be continuous • event tasks: supports consumed tag trigger and EVENT instruction	• 32 tasks (only 1 continuous) • event tasks: supports all event triggers	• 8 tasks (only 1 continuous) • event tasks: supports consumed tag trigger and EVENT instruction	• 8 tasks (only 1 continuous) • event tasks: supports axis and motion event triggers
user memory	1756-L55M12 750 Kbytes 1756-L55M13 1.5 Mbytes 1756-L55M15 3.5 Mbytes 1756-L55M16 7.5 Mbytes 1756-L55M22 750 Kbytes 1756-L55M23 1.5 Mbytes 1756-L55M24 3.5Mbytes 1756-L61 2 Mbytes 1756-L62 4 Mbytes 1756-L63 8 Mbytes	1769-L20 64 Kbytes 1769-L30 256 Kbytes 1769-L31 512 Kbytes 1769-L32E 750 Kbytes 1769-L35E 1.5 Mbytes	1789-L10 2 Mbytes 3 slots, no motion 1789-L30 64 Mbytes 5 slots 1789-L60 64 Mbytes 16 slots	1794-L33 64 Kbytes 1794-L34 512 Kbytes	256 Kbytes 768 Kbytes with expansion memory
nonvolatile user memory	1756-L55M12 none 1756-L55M13 none 1756-L55M15 none 1756-L55M16 none 1756-L55M22 yes 1756-L55M23 yes 1756-L55M24 yes 1756-L6x CompactFlash	1769-L20 yes 1769-L30 yes 1769-L31 CompactFlash 1769-L32E CompactFlash 1769-L35E CompactFlash	none	1794-L33 yes 1794-L34/B yes	yes (expansion memory)
built-in communication ports	1 port RS-232 serial (DF1 or ASCII)	• 1769-L20: 1 RS-232 serial port (DF1 or ASCII) • 1769-L3, -L31: 2 RS-232 serial ports (one DF1 only, other DF1 or ASCII) • 1769-L32E, -L35E: 1 EtherNet/IP port and 1 RS-232 serial port (DF1 or ASCII)	depends on personal computer	• 1 RS-232 serial port (DF1 or ASCII) • 2 slots for 1788 communication cards	• 1 RS-232 serial port (DF1 or ASCII) • 1 slot for 1788 communication cards
communication options (these options have specific products and profiles for their platform - other options are available via 3rd party products and generic profiles)	EtherNet/IP ControlNet DeviceNet Data Highway Plus Universal Remote I/O serial Modbus via ladder routine DH-485 SynchLink	EtherNet/IP DeviceNet serial Modbus via ladder routine DH-485	EtherNet/IP ControlNet DeviceNet serial	EtherNet/IP ControlNet DeviceNet serial Modbus via ladder routine DH-485	EtherNet/IP ControlNet DeviceNet serial Modbus via ladder routine DH-485
integrated motion support	SERCOS interface analog interface hydraulic interface	not applicable	SERCOS interface analog interface	not applicable	1 full servo 1 feedback axis
mounting and /or installation options	1756 chassis	panel mount DIN rail	none	panel mount DIN rail	embedded in PowerFlex 700S
programming languages	• relay ladder • structured text • function block • sequential function chart	• relay ladder • structured text • function block • sequential function chart	• relay ladder • structured text • function block • sequential function chart • external routines	• relay ladder • structured text • function block • sequential function chart	• relay ladder • structured text • function block • sequential function chart

Designing Logix5000 Systems

Introduction

This reference manual provides guidelines you can follow to optimize your system. This manual also provides system information you need to make system design choices. As you read this manual:

This symbol:	Indicates:
	guidelines you should follow programming practices that can improve system performance things you can do
	considerations you should know when making design choices system information that can affect system performance things you should know

In addition to the controller-specific topics covered in each chapter, the back of this manual includes a:

- glossary of commonly used terms
- list of related publications

This manual is meant for experienced Logix-system programmers. The information in this manual is presented with the assumption that the reader understands how to implement the guidelines. The list of related publications at the back of the manual identifies resources you can use for more details on how to implement the guidelines.

Important User Information

Solid state equipment has operational characteristics differing from those of electromechanical equipment. *Safety Guidelines for the Application, Installation and Maintenance of Solid State Controls* (Publication SGI-1.1 available from your local Rockwell Automation sales office or online at http://www.ab.com/manuals/gi) describes some important differences between solid state equipment and hard-wired electromechanical devices. Because of this difference, and also because of the wide variety of uses for solid state equipment, all persons responsible for applying this equipment must satisfy themselves that each intended application of this equipment is acceptable.

In no event will Rockwell Automation, Inc. be responsible or liable for indirect or consequential damages resulting from the use or application of this equipment.

The examples and diagrams in this manual are included solely for illustrative purposes. Because of the many variables and requirements associated with any particular installation, Rockwell Automation, Inc. cannot assume responsibility or liability for actual use based on the examples and diagrams.

No patent liability is assumed by Rockwell Automation, Inc. with respect to use of information, circuits, equipment, or software described in this manual.

Reproduction of the contents of this manual, in whole or in part, without written permission of Rockwell Automation, Inc. is prohibited.

Throughout this manual we use notes to make you aware of safety considerations.

WARNING

Identifies information about practices or circumstances that can cause an explosion in a hazardous environment, which may lead to personal injury or death, property damage, or economic loss.

IMPORTANT

Identifies information that is critical for successful application and understanding of the product.

ATTENTION

Identifies information about practices or circumstances that can lead to personal injury or death, property damage, or economic loss. Attentions help you:

- identify a hazard
- avoid a hazard
- recognize the consequence

SHOCK HAZARD

Labels may be located on or inside the drive to alert people that dangerous voltage may be present.

BURN HAZARD

Labels may be located on or inside the drive to alert people that surfaces may be dangerous temperatures.

Table of Contents

Chapter 1

Logix5000 Controller Resources

Introduction 1-1
Using Connections for Communications 1-3
Determining Total Connection Requirements 1-5

Chapter 2

Dividing Logic into Tasks, Programs, and Routines

Introduction 2-1
Deciding When to Use Tasks, Programs, and Routines 2-2
Specifying Task Priorities 2-3
Managing User Tasks 2-5
Factors that Affect Task Execution 2-6
Configuring a Continuous Task 2-8
Configuring a Periodic Task 2-8
Configuring an Event Task 2-8
Guidelines for Configuring an Event Task 2-9
Selecting a System Overhead Percentage 2-10
Managing the System Overhead Timeslice Percentage 2-11
Developing Application Code in Routines 2-12
Programming Methods 2-13
Controller Prescan of Logic 2-14
Controller Postscan of SFC Logic 2-14

Chapter 3

Addressing Data

Introduction 3-1
Guidelines for Data Types 3-2
Arrays 3-3
Guidelines for Arrays 3-4
Indirect Addressing of Arrays 3-5
Guidelines for Array Indexes 3-6
Prescan of an Array Index 3-6
Guidelines for User-Defined Structures 3-7
Selecting a Data Type for Bit Tags 3-8
Serial Bit Addressing 3-9
Guidelines for String Data Types 3-10
PLC-5/SLC 500 Access of Strings 3-10
Configuring Tags 3-11
Guidelines for Base Tags 3-12
Creating Alias Tags 3-13
Guidelines for Data Scope 3-13

Chapter 4

Sharing Tag Data with Other Controllers (Produced and Consumed Tags)

Introduction 4-1
Guidelines for Creating Produced and Consumed Tags 4-2
Guidelines for Specifying an RPI Rate 4-3
Guidelines for Managing Connections for Produced and Consumed Tags 4-3
Configuring an Event Task Based on a Consumed Tag 4-3
Comparing Messages and Produced/Consumed Tags 4-4

Chapter 5

Designing Networks

Introduction 5-1
Select a Network 5-1
EtherNet/IP Network Topology 5-2
Guidelines for EtherNet/IP 5-3
ControlNet Network Topology 5-4
Guidelines for ControlNet 5-5
DeviceNet Network Topology 5-6
Guidelines for DeviceNet 5-7

Chapter 6

Communicating with I/O

Introduction 6-1
Buffering I/O Data 6-1
Guidelines for Specifying an RPI Rate for I/O Modules 6-2
Communication Formats for I/O Modules 6-3
Guidelines for Managing I/O Connections 6-4
Guidelines for Managing I/O Connections (continued) 6-5
Creating Tags for I/O Data 6-6
Controller Ownership 6-7

Chapter 7

Communicating with Other Devices

Introduction 7-1
Caching Messages 7-2
Message Buffers 7-2
Guidelines for Messages 7-4
Guidelines for Managing Message Connections 7-5
Guidelines for Block-Transfer Messages 7-6
Mapping Tags 7-6

Chapter 8

Optimizing an Application for Motion Control

Introduction 8-1
Coarse Update Rate 8-1
Axis Limits 8-2
Performance Limits 8-2
Motion Event Task Triggers 8-3

Chapter 9

Optimizing an Application for Use with HMI

Introduction 9-1
Guidelines for HMI Applications 9-2
Comparison of RSView32 and RSView Enterprise 9-2
How RSLinx Software Communicates with Logix5000 Controllers 9-3
Guidelines for RSLinx Software 9-4
Comparison of RSLinx Classic and RSLinx Enterprise 9-5
Guidelines for Configuring Controller Tags 9-6

Chapter 10

Optimizing an Application for Process Control

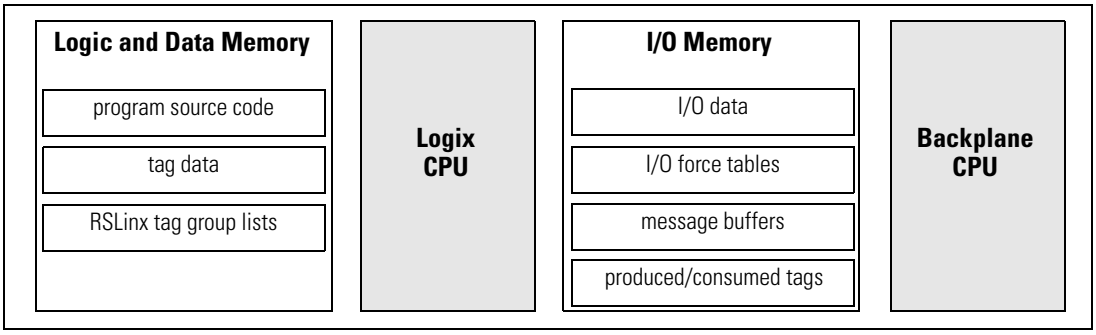
Introduction 10-1
Comparison of PID and PIDE Instructions 10-1
Guidelines for Programming PID Loops 10-2
Advanced Process Instructions 10-3
Faceplates 10-3

Logix5000 Controller Resources

Introduction

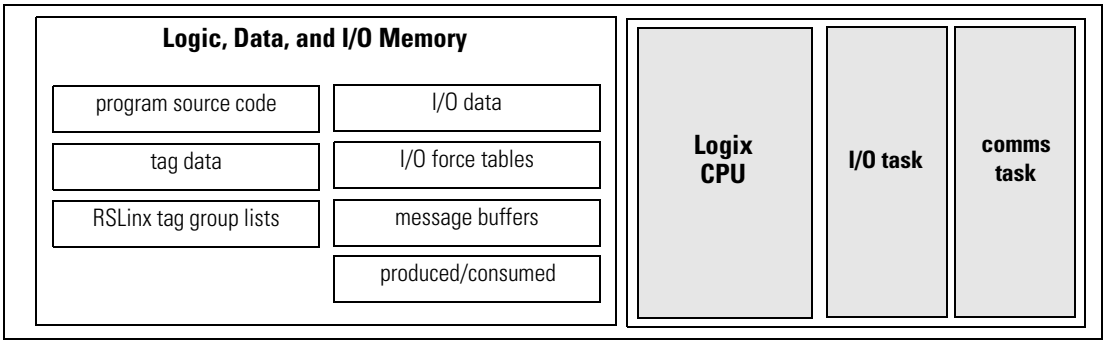
Depending on the controller, resources are divided differently:

ControlLogix controllers - memory is separated into two, isolated sections



- The Logix CPU executes application code and messages.
- The backplane CPU communicates with I/O and sends/receives data from the backplane. This CPU operates independently from the Logix CPU, so it sends and receives I/O information asynchronous to program execution.

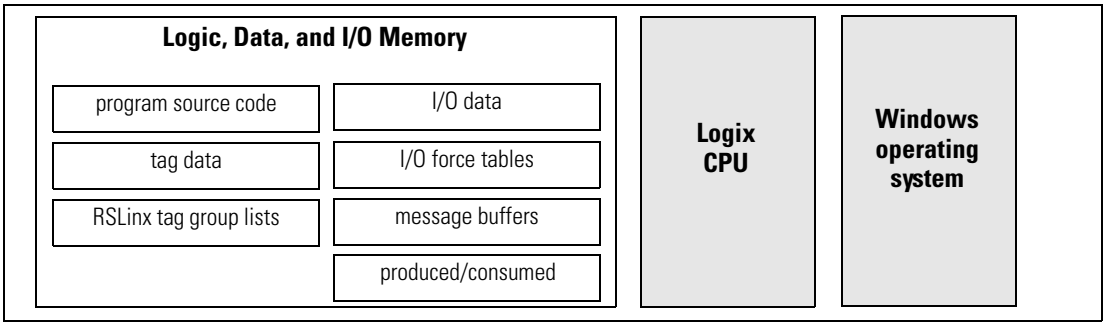
CompactLogix, FlexLogix, and DriveLogix controllers - memory is in one, contiguous section



These controllers have a single CPU that performs all operations. Isolated tasks perform I/O and communications and interact with networks. These tasks simulate the backplane CPU.

For this controller:	The I/O task is priority:	The communications task is priority:
CompactLogix, FlexLogix, and DriveLogix	6	12

SoftLogix controllers - memory is in one, contiguous section



The SoftLogix controller has a single CPU that works in conjunction with the Windows operating system to perform all operations. Rather than using controller priority levels for I/O and communications tasks, the SoftLogix controller uses Windows priority levels for these tasks.

For this controller:	The I/O task is:	The communications task is:
SoftLogix	Windows priority 16 (Idle)	Windows priority 16 (Idle)

For all controllers, memory is used at run time for:

- message processing to buffer incoming and outgoing messages
- RSLinx data handling to store tag groups
- online edits to store edit rungs
- graphical trends to buffer data

Estimating memory use

The following equations provide an estimate of the memory needed for a controller.

Controller tasks	_____ * 4000	=	_____ bytes (minimum 1 needed)
Discrete I/O points	_____ * 400	=	_____ bytes
Analog I/O points	_____ * 2600	=	_____ bytes
DeviceNet modules[1]	_____ * 7400	=	_____ bytes
Other communication modules[2]	_____ * 2000	=	_____ bytes
Motion axis	_____ * 8000	=	_____ bytes
	Total	=	_____ bytes

[1]The first DeviceNet module is 7400 bytes. Additional DeviceNet modules are 5800 bytes each.

[2]Count all the communication modules in the system, not just those in the local chassis. This includes device connection modules, adapter modules, and ports on PanelView terminals.

Reserve 20-30% of the controller memory to accommodate growth.

RSLinx use of Logix5000 controller memory

The amount of memory RSLinx needs depends on the type of data RSLinx reads. The following equations provide an estimate of the memory needed for RSLinx communications.

RSLinx overhead (per connection)	_____	* 1345 =	_____	bytes (4 connections by default)
Individual tags	_____	* 45 =	_____	bytes
Arrays / structures	_____	* 7 =	_____	bytes
		Total =	_____	bytes

Consolidating tags into an array or a structure reduces the communications overhead and the number of connections needed to obtain the data.

PLC/SLC memory comparison

The Logix5000 controllers used compiled instructions to provide faster execution times than PLC or SLC processors. The compiled instructions use more memory when compared to the instructions in PLC and SLC processors.

If you have a PLC/SLC program, you can estimate the number of bytes it will take in a Logix5000 controller by:

number PLC/SLC words * 18 = *number of Logix5000 bytes*

Using Connections for Communications

A Logix5000 controller uses a connection to establish a communication link between two devices. Connections can be:

- controller to local I/O modules or local communication modules
- controller to remote I/O or remote communication modules
- controller to remote I/O (rack optimized) modules

  For more information on connections for I/O, see Chapter 6 "*Communicating with I/O*."
- produced and consumed tags

  For more information, see Chapter 4, "*Sharing Tag Data with Other Controllers*."
- messages

  For more information, see Chapter 7 "*Communicating with Other Devices*."
- access to RSLogix 5000 programming software
- RSLinx software access for HMI or other software applications

These controllers support:

Controller:	Number of Connections:
ControlLogix SoftLogix	250
1769-L31, -L32E, -L35E CompactLogix DriveLogix FlexLogix	100
1769-L20, -L30 CompactLogix	17

The limit of connections may ultimately reside in the communication module you use for the connection. If a message path routes through a communication module, the connection related to the message also counts towards the connection limit of that communication module.

For this controller:	This communication device:	Supports this number of connections:
ControlLogix	1756-CNB	64 connections depending on RPI, recommend using only 48 connections (any combination of scheduled and message connections)
	1756-ENBT	128 connections (all connections are message connections)
CompactLogix	1769-L32E, -L35E	32 connections (over EtherNet/IP only)
FlexLogix PowerFlex 700S with DriveLogix	1788-CNx, -CNxR	32 connections depending on RPI, as many as 22 connections can be scheduled The remaining connections (or all 32, if you have no scheduled connections) can be used for message connections
	1788-ENBT	32 connections (all 32 connections are message connections)
SoftLogix5800	1784-PCICS	128 connections 127 of which can be scheduled connections

Determining Total Connection Requirements

The total connections for a Logix5000 controller include both local and remote connections. Tallying local connections is not an issue for FlexLogix or CompactLogix controllers because both support the maximum number of modules allowed in their systems. The ControlLogix and SoftLogix controllers support more communication modules than the other controllers, so you must tally local connections to make sure you stay within the 250 connection limit. Use this table to tally **local** connections.

Connection Type:	Device Quantity:	x	Connections per Module:	=	Total Connections:
local I/O module (always a direct connection)		x	1	=	
motion servo module		x	3	=	
ControlNet communication module		x	0	=	
EtherNet/IP communication module		x	0	=	
DeviceNet communication module		x	2	=	
DH+/Remote I/O communication module		x	1	=	
RSLogix 5000 programming software access to controller		x	1	=	
total					

The communication module(s) you select determines how many remote connections are available for I/O and information. Use this table to tally **remote** connections:

Connection Type:	Device Quantity:	x	Connections per Module:	=	Total Connections:
remote ControlNet communication module configured as a direct (none) connection configured as a rack-optimized connection		x	 0 **or** 1	=	
distributed I/O module over ControlNet (direct connection)		x	1	=	
remote EtherNet/IP communication module configured as a direct (none) connection configured as a rack-optimized connection		x	 0 **or** 1	=	
distributed I/O module over EtherNet/IP (direct connection)		x	1	=	
remote device over DeviceNet (accounted for in rack-optimized connection for local DeviceNet module)		x	0	=	
other remote communication adapter		x	1	=	
produced tag and first consumer each additional consumer		x	1 1	=	
consumed tag		x	1	=	
connected message (CIP Data Table Read/Write and DH+)		x	1	=	
block-transfer message		x	1	=	
RSLinx software access for HMI or other software applications		x	4	=	
RSLinx Enterprise software for HMI or other software applications		x	5	=	
total					

Notes:

Dividing Logic into Tasks, Programs, and Routines

Introduction

The controller operating system is a preemptive multitasking system that is IEC 61131-3 compliant. This environment provides:

tasks to configure controller execution

A task provides scheduling and priority information for a set of one or more programs. You can configure tasks as either continuous, periodic, or event.

programs to group data and logic

A task can have as many as 32 programs, each with its own routines and program-scoped tags. Once a task is triggered (activated), all the programs assigned to the task execute in the order in which they are listed in the Controller Organizer.

Programs are useful for projects developed by multiple programmers. During development, the code in one program that makes use of program-scoped tags, can be duplicated in a second program and minimize the possibility of tag names colliding.

routines to encapsulate executable code written in a single programming language

Routines contain the executable code. Each program has a main routine that is the first routine to execute within a program. Use logic, such as the Jump to Subroutine (JSR) instruction, to call other routines. You can also specify an optional program fault routine.

See *"Developing Application Code in Routines"* on page 2-12 for information on selecting programming languages and how the controller prescans and postscans logic.

Deciding When to Use Tasks, Programs, and Routines

Use these considerations to determine when to use a task, program or routine:

Comparison:	Task:	Program:	Routine:
Quantity available	varies by controller (4, 8, or 32)	32 programs per task	unlimited number of routines per program
Function	determines how and when code will be executed	organizes groups of routines that need to share a common data area	contains executable code (relay ladder, function block diagram, sequential function chart, or structured text) that controls the machine
Use	• most code should reside in a continuous task • use a periodic task for slower processes or when time-based operation is critical • use an event task for operations that require synchronization to a specific event	• put major equipment pieces or plant cells into isolated programs • use programs to isolate different programmers or create reusable code • configurable execution order within a task	• isolate machine or cell functions in a routine • use the appropriate language for the process • modularize code into subroutines that can be called multiple times
Considerations	• a high number of tasks can be difficult to debug • may need to disable output processing on some tasks to improve performance • tasks can be inhibited to prevent execution	• data spanning multiple programs must go into controller-scoped area • listed in the Controller Organizer in the order of execution	• subroutines with multiple calls can be difficult to debug • data can be referenced from program-scoped and controller-scoped areas • calling a large number of routines impacts scan time • listed in the Controller Organizer as Main, Fault, and then alphabetically

Specifying Task Priorities

Each task in the controller has a priority level that determines which task executes when multiple tasks are triggered. A higher priority task (such as 1) interrupts any lower priority task (such as 15). The continuous task has the lowest priority and is always interrupted by a periodic or event task.

This Logix5000 controller:	Supports this many user tasks:	And has this many priority levels:
ControlLogix	32	15
1769-L35E CompactLogix	8	15
1769-L32E CompactLogix	6	15
1769-L20, -L30, -L31 CompactLogix	4	15
FlexLogix	8	15
PowerFlex 700A with DriveLogix	8	15
SoftLogix5800	32	3

If a periodic or event task is executing when another is triggered and both tasks are at the same priority level, the tasks timeslice execution time in 1 msec increments until one of the tasks completes execution

The Logix5000 controller has these types of tasks.

Priority:	User Task:	Description:
Highest	na	CPU overhead - serial port and general CPU operations
	na	Motion planner - executed at coarse update rate
	na	Redundancy task - communications to 1757-SRM in redundant systems
	na	Trend data collection - high-speed collection of trend data values
	Priority 1 Event/Periodic	na
	Priority 2 Event/Periodic	na
	Priority 3 Event/Periodic	na
	Priority 4 Event/Periodic	na
	Priority 5 Event/Periodic	na
	Priority 6 Event/Periodic	CompactLogix and FlexLogix controllers process I/O as a periodic task based on the chassis RPI setting
	Priority 7 Event/Periodic	na
	Priority 8 Event/Periodic	na
	Priority 9 Event/Periodic	na
	Priority 10 Event/Periodic	na
	Priority 11 Event/Periodic	na
	Priority 12 Event/Periodic	DriveLogix communications to drives. CompactLogix and FlexLogix communications and scheduled connection maintenance
	Priority 13 Event/Periodic	na
	Priority 14 Event/Periodic	na
	Priority 15 Event/Periodic	na
Lowest	Continuous	Message handler - based on system overhead timeslice

Managing User Tasks

You can configure these user task:

If you want logic to execute:	Then use a:	Description:
all of the time	continuous task	The continuous task runs in the background. Any CPU time not allocated to other operations or tasks is used to execute the continuous task. • The continuous task runs all the time. When the continuous task completes a full scan, it restarts immediately. • A project does not require a continuous task. If used, there can be only one continuous task.
• at a constant period (e.g., every 100 ms) • multiple times within the scan of your other logic	periodic task	A periodic task performs a function at a specific time interval. Whenever the time for the periodic task expires, the periodic task: • interrupts any lower priority tasks • executes one time • returns control to where the previous task left off
immediately when an event occurs	event task	An event task performs a function only when a specific event (trigger) occurs. Whenever the trigger for the event task occurs, the event task: • interrupts any lower priority tasks • executes one time • returns control to where the previous task left off See "*Configuring an Event Task*" on page 2-8 for the triggers for an event task. Some Logix5000 controllers do not support all triggers.

The user tasks you create show up in the Tasks folder of the controller. These pre-defined, system tasks do not show up in the Tasks folder and they do not count toward the task limit of the controller:

- motion planner
- I/O processing
- system overhead
- output processing

Factors that Affect Task Execution

motion planner

See also "Optimizing an Application for Motion Control" on page 8-1.

The motion planner interrupts all other tasks, regardless of their priority.

- The number of axes and coarse update period for the motion group affect how long and how often the motion planner executes.
- If the motion planner is executing when a task is triggered, the task waits until the motion planner is done.
- If the coarse update period occurs while a task is executing, the task pauses to let the motion planner execute.

I/O processing

CompactLogix, FlexLogix, DriveLogix, and SoftLogix controllers use a dedicated periodic task to process I/O data. This I/O task:

- For CompactLogix, FlexLogix, and DriveLogix, operates at priority 6. For SoftLogix, operates at Windows priority 16 (Idle).
- Higher-priority tasks take precedence over the I/O task and can impact processing.
- Executes at the fastest RPI you have scheduled for the system.
- Executes for as long as it takes to scan the configured I/O modules.
- For local I/O, updates also occur at the end of each task.

system overhead

See also "Selecting a System Overhead Percentage" on page 2-10.

System overhead is the time that the controller spends on message communication and background tasks.

- Message communication is any communication that you do *not* configure through the I/O configuration folder of the project, such as MSG instructions.
- Message communication occurs only when a periodic or event task is not running. If you use multiple tasks, make sure that their scan times and execution intervals leave enough time for message communication.
- System overhead interrupts only the continuous task.
- The system overhead time slice specifies the percentage of time (excluding the time for periodic or event tasks) that the controller devotes to message communication.
- The controller performs message communication for up to 1 ms at a time and then resumes the continuous task.
- Adjust the update rates of the tasks as needed to get the best trade-off between executing your logic and servicing message communication.

output processing

At the end of a task, the controller performs overhead operations (output processing) for the output modules in your system. This output processing may effect the update of the I/O modules in your system. You can turn off output processing for a specific task, which reduces the elapsed time of that task.

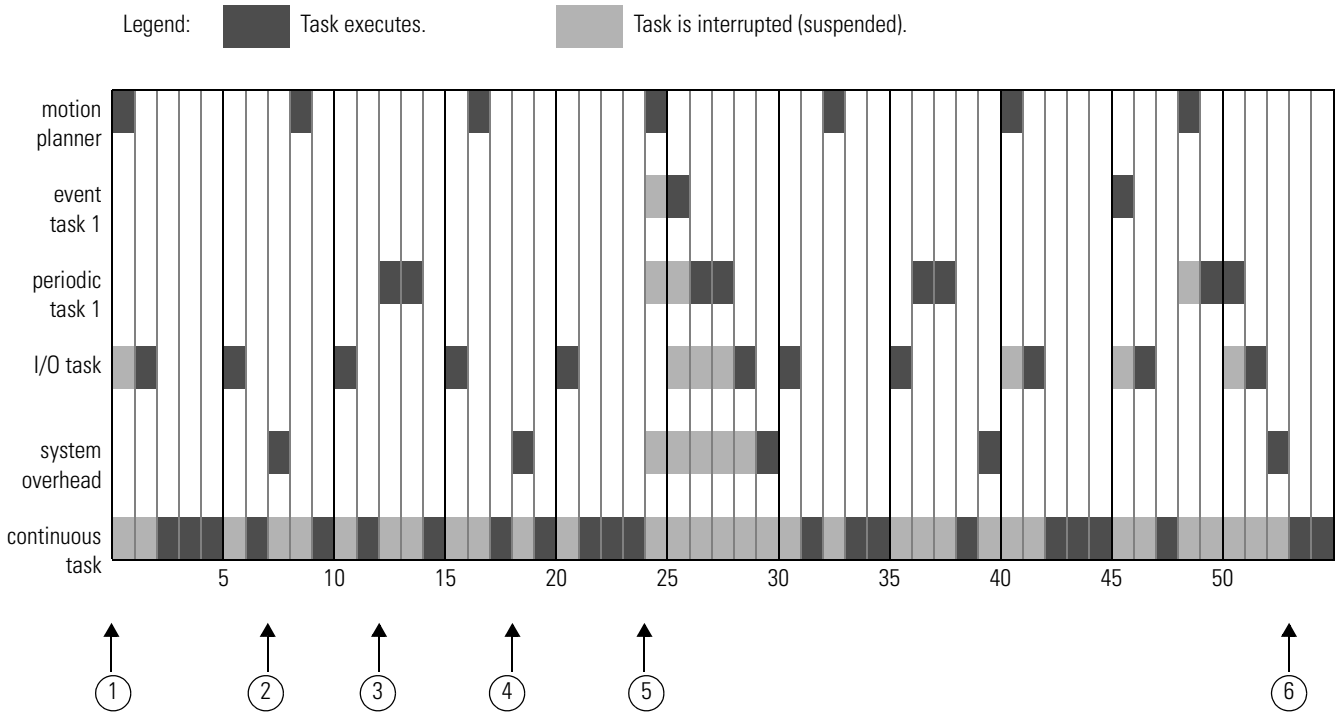
too many tasks

If you have too many tasks, then:

- The continuous task may take too long to complete.
- Other tasks may experience overlaps. If a task is interrupted too frequently or too long, it may not complete its execution before it is triggered again.
- Controller communications might be slower.
- If your application is designed for data collection, try to avoid multiple tasks. Switching between multiple tasks limits communication bandwidth.

The following example depicts the execution of a project with these tasks:

Task:	Priority:	Period:	Execution time:	Duration:
motion planner	n/a	8 ms (course update rate)	1 ms	1 ms
event task 1	1	n/a	1 ms	1 to 2 ms
periodic task 1	2	12 ms	2 ms	2 to 4 ms
I/O task—n/a to ControlLogix and SoftLogix controllers	7	5 ms (fastest RPI)	1 ms	1 to 5 ms
system overhead	n/a	time slice = 20%	1 ms	1 to 6 ms
continuous task	n/a	n/a	20 ms	48 ms



Description:	
1	Initially, the controller executes the motion planner and the I/O task (if one exists).
2	After executing the continuous task for 4 ms, the controller triggers the system overhead.
3	The period for periodic task 1 expires (12 ms), so the task interrupts the continuous task.
4	After executing the continuous task again for 4 ms, the controller triggers the system overhead.
5	The triggers occur for event task 1. Event task 1 waits until the motion planner is done. Lower priority tasks experience longer delays.
6	The continuous task automatically restarts.

Configuring a Continuous Task

The continuous task is created automatically when you open an RSLogix 5000 project. A continuous task is similar to how logic executes on PLC-5 and SLC 500 processors. A Logix5000 controller supports one continuous task, but a continuous task is not required. You can configure whether the task updates output modules at the end of the continuous task. You can change the continuous task to either a periodic or event task.

The CPU timeslices between the continuous task and system overhead. Each task switch between user task and system overhead takes additional CPU time to load and restore task information. See "*Selecting a System Overhead Percentage*" on page 2-10.

Configuring a Periodic Task

A periodic task executes automatically based on a preconfigured interval. This task is similar to selectable timed interrupts in PLC-5 and SLC 500 processors. You can configure whether the task updates output modules at the end of the periodic task. After the task executes, it does not execute again until the configured time interval has elapsed.

If your application has a lot of communications (such as message instructions or RSLinx communications), use a periodic task rather than a continuous task. This avoids the overhead associated with task switching, which can improve performance.

Configuring an Event Task

An event task executes automatically based on a preconfigured event occurring. You can configure whether the task updates output modules at the end of the task. After the task executes, it does not execute again until the configured event occurs again. Each event task requires a specific trigger that defines when the task is to execute. You can select from:

Use this trigger:	Description:
Module Input Data State Change	The input module (digital or analog) triggers the event task based on the change of state (COS) configuration for the module. Enable COS for only one point on the module. If you enable COS for multiple points, a task overlap of the event task may occur. The ControlLogix sequence of events modules (1756-IB16ISOE, 1756-IH16ISOE) use the Enable CST Capture feature instead of COS.
Consumed Tag	Only one consumed tag can trigger a specific event task. Use an IOT instruction in the producing controller to signal the production of new data.
Axis Registration 1 or 2	A registration input triggers the event task.
Axis Watch	A watch position triggers the event task.
Motion Group Execution	The coarse update period for the motion group triggers the execution of both the motion planner and the event task. Because the motion planner interrupts all other tasks, it executes first.
EVENT instruction	Multiple EVENT instructions can trigger the same task.

For more information on event tasks, see:

- *Logix5000 Controllers Common Procedures Programming Manual*, publication 1756-PM001
- *Using Event Tasks with Logix5000 Controllers*, LOGIX-WP003

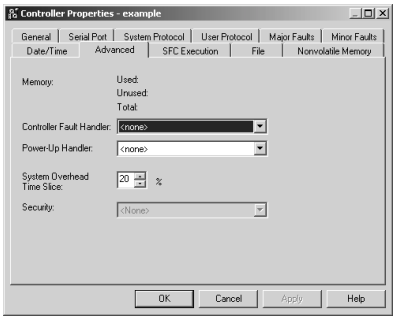
Guidelines for Configuring an Event Task

Place the I/O module being used to trigger an event in the same chassis as the controller	Placing the I/O module in a remote chassis adds additional network communications and processing to the response time.
Limit events on digital inputs to a single input bit on a module	All inputs on a module trigger a single event, so using multiple bits increases the chance of a task overlap. Configure the module to detect change-of-state on the trigger input and turn off the other bits.
Set the priority of the event task as the highest priority on the controller	If the priority of the event task is lower than a periodic task, the event task will have to wait for the periodic task to complete execution.
Limit the number of event tasks	Increasing the number of event tasks reduces the available CPU bandwidth and increases the chances of task overlap.

Additional considerations for event tasks

Consideration:	Description:
amount of code in the event task	Each logic element (rung, instruction, structured text construct, etc…) adds to scan time.
task priority	If the event task is not the highest priority task, a higher priority task may delay or interrupt the execution of the event task.
CPS and UID instructions	If one of these instructions are active, the event task cannot interrupt the currently executing task. (The task with the CPS or UID.)
communication interrupts	The following actions interrupt a task, regardless of the priority of the task: • arrival of scheduled module and consumed tag information via the backplane • serial port communication
motion planner	The motion planner takes precedence over an event task.
trends	Trend data collection takes precedence over an event task.

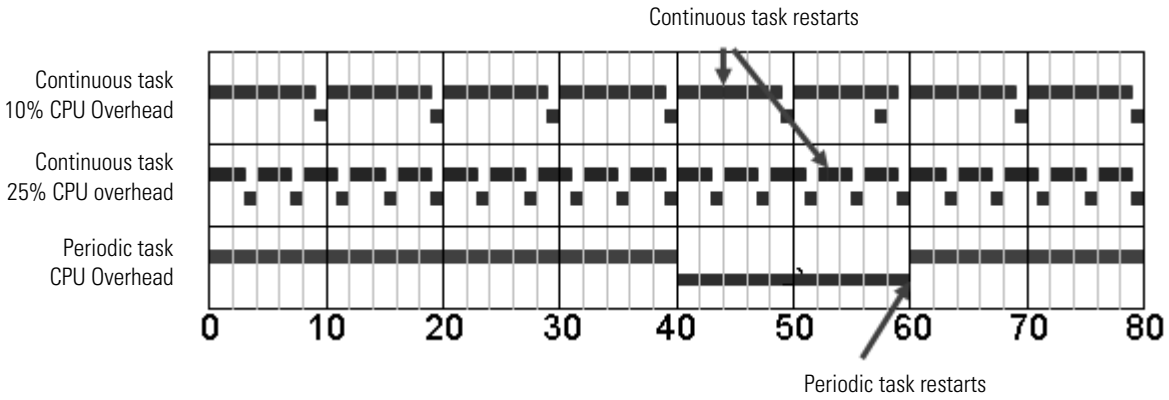
Selecting a System Overhead Percentage



The system overhead timeslice specifies the percentage of continuous task execution time that is devoted to communication and background functions. System overhead functions include:

- communicating with programming and HMI devices (such as RSLogix 5000 software)
- responding to messages
- sending messages
- serial port message and instruction processing

The controller performs system overhead functions for up to 1 ms at a time. If the controller completes the overhead functions in less than 1 ms, it resumes the continuous task. The following chart compares a continuous and periodic task.



Example:	Description:
Continuous task 10% CPU overhead	In the top example, the system overhead timeslice is set to 10%. Given 40 msec of code to execute, the continuous task completes the execution in 44 msec. During a 60 msec timespan, the controller is able to spend 5 msec on communications processing.
Continuous task 25% CPU overhead	By increasing the system overhead timeslice to 25%, the controller completes the continuous task scan in 57 msec and spends 15 msec of a 60 msec timespan on communications processing.
Periodic task	Placing the same code in a periodic task yields even more time for communications processing. The bottom example assumes the code is in a 60 msec periodic task. The code executes to completion and the goes dormant until the 60 msec, time-based trigger occurs. While the task is dormant, all CPU bandwidth can focus on communications. Since the code only takes 40 msec to execute, the controller can spend 20 msec on communications processing. Depending on the amount of communications to process during this 20 msec window, it can be delayed as it waits for other modules in the system to process all the data that was communicated.

The Logix5000 CPU timeslices between the continuous task and system overhead. Each task switch between user task and system overhead takes additional CPU time to load and restore task information. You can calculate the continuous task interval as:

```
ContinuousTime=(100/SystemOverheadTimeSlice%) - 1
```

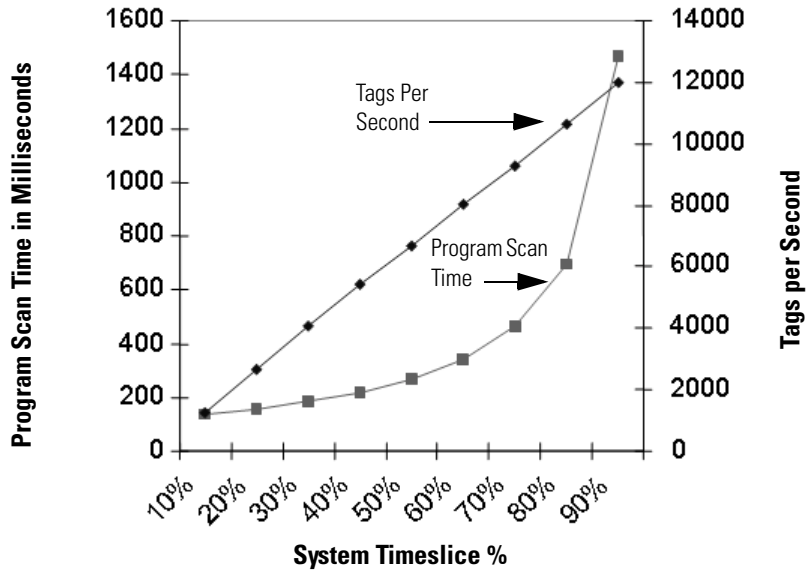
Managing the System Overhead Timeslice Percentage

As the system overhead timeslice percentage increases, time allocated to executing the continuous task decreases. If there are no communications for the controller to manage, the controller uses the communications time to execute the continuous task.

impact on communications and scan time

Increasing the system overhead timeslice percentage decreases execution time for the continuous task while it increases communications performance.

Increasing the system overhead timeslice percentage also increases the amount of time it takes to execute a continuous task - increasing overall scan time.



Individual applications may differ, but the overall impact on communications and scan time remains the same. The above data is based on a ControlLogix5555 controller running a continuous task with 5000 tags (no arrays or user-defined structures).

Developing Application Code in Routines

Each routine contains logic in one programming language. Choose a programming language based on the application

In general, if a section of your code represents:	Then use this language:
continuous or parallel execution of multiple operations (not sequenced)	ladder logic
boolean or bit-based operations	
complex logical operations	
message and communication processing	
machine interlocking	
operations that service or maintenance personnel may have to interpret in order to troubleshoot the machine or process.	
servo motion control	
continuous process and drive control	function block diagram
loop control	
calculations in circuit flow	
high-level management of multiple operations	sequential function chart (SFC)
repetitive sequences of operations	
batch process	
motion control sequencing (via sequential function chart with embedded structure text)	
state machine operations	
complex mathematical operations	structured text
specialized array or table loop processing	
ASCII string handling or protocol processing	

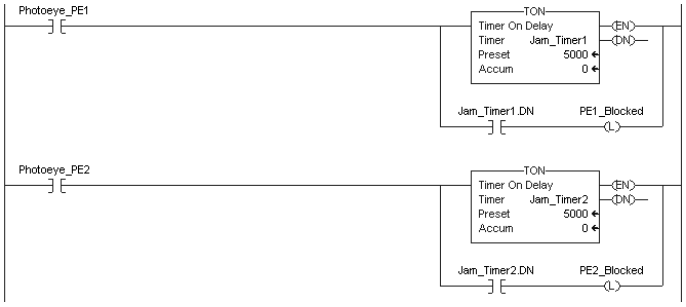
Programming Methods

The capabilities of the Logix5000 controllers make different programming methods possible. There are tradeoffs to consider when selecting a programming method:

Inline duplication

- uses more memory
- fastest execution time because all tag references are defined before run time
- easiest to maintain because rung animation matches tag values
- requires more time to create and modify

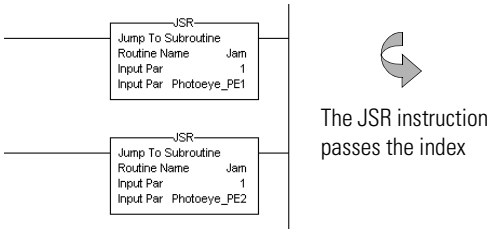
Write multiple copies of the code with different tag references.



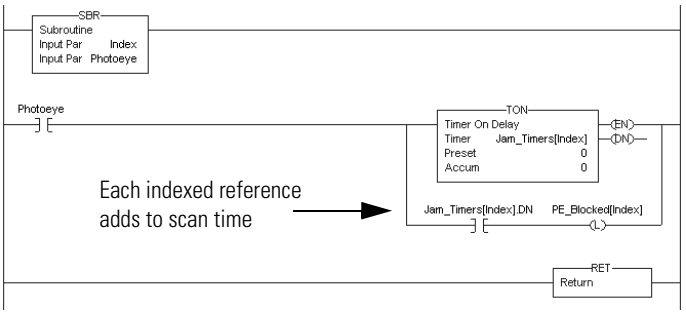
Indexed routine

- one copy of code is faster to develop
- slowest execution time because all tag references are calculated at run time
- can be difficult to maintain because the data monitor is not synchronized to execution

Write one copy of code and use indexed references to data stored in arrays.



The JSR instruction passes the index

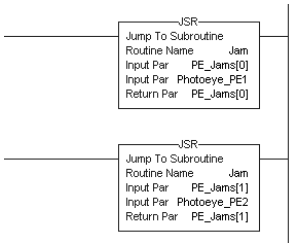


Buffered routine

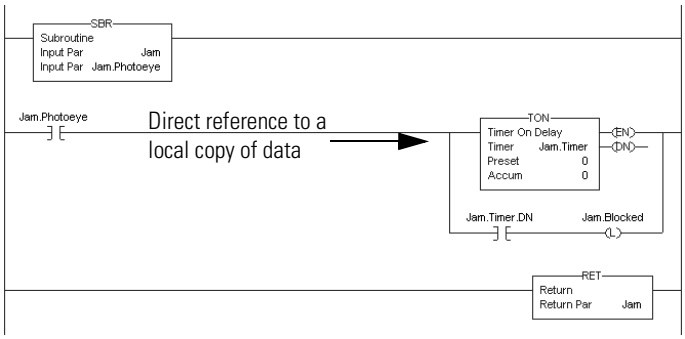
- one copy operation can occur faster than multiple index offsets
- eliminates the need to calculate array offsets at run time
- the amount of code increases, but so do the benefits
- can be difficult to maintain because the data monitor is not synchronized to execution

Copy the values of an array into tags and reference these buffer tags directly.

A user-defined structure consolidates control data



The JSR instruction passes all control instance data



Controller Prescan of Logic

On power-up, the controller prescans logic to initialize instructions. The controller resets all state-based instructions, such as outputs (OTE) and timers (TON). Some instructions also perform operations during prescan. For example, the ONSR instructions turns off the storage bit. For information on prescan, see:

- *Logix5000 Controllers General Instructions Reference Manual*, publication 1756-RM003
- *Logix5000 Controllers Common Procedures Programming Manual*, publication 1756-PM001
- *Logix5000 Controllers Process Control and Drives Instructions Reference Manual*, publication 1756-RM006

During prescan, input values are not current and outputs are not written.

affects of prescan on relay ladder logic	The controller resets non-retentive I/O and internal values.
affects of prescan on function block diagram logic	In addition to resetting non-retentive I/O and internal values, the controller clears the EnableIn parameter for every function block diagram.
affects of prescan on structured text logic	The controller resets bit tags and forces numeric tags to zero (0). Use the bracketed assignment operator ([:=]) to force a value to be reset during prescan. If you want a tag left in its last state, use the non-bracketed assignment operator (:=).
affects of prescan on sequential function chart logic	Embedded structured text follows the same rules as listed above.
affects of prescan on array indexed values	Array index values can fault the controller during prescan. If an array index value is larger than the dimension of the array, the controller will detect a major fault during prescan. To avoid this, make sure the index is always set properly or use a fault routine to handle this error during prescan. See "*Prescan of an Array Index*" on page 3-6.

Controller Postscan of SFC Logic

SFCs support an automatic reset option that performs a postscan of the actions associated with a step once a transition indicates that the step is completed. Also, every Jump to Subroutine (JSR) instruction causes the controller to postscan the called routine. During this postscan:

- output energize (OTE) instructions are turned off and non-retentive timers are reset.
- in structured text code, use the bracketed assignment operator ([:=]) to have tags reset
- in structured text code, use the non-bracketed assignment operator (:=) to have tags left in their last state.

Addressing Data

Introduction

Logix5000 controllers support IEC 61131-3 atomic data types, such as BOOL, SINT, INT, DINT, and REAL. The controllers also support compound data types, such as arrays, predefined structures (such as counters and timers) and user-defined structures.

atomic data type (BOOL, SINT, INT, DINT, REAL)

Benefits:	Considerations:
• individual names • no limit to the number of tags • Tag Editor and Data Monitor can filter individual tags and display any references • always listed alphabetically in the Tag Editor and Data Monitor • full alias tag support (both the base tag and its bits) • can be added when programming online	• each tag uses 32 bits of memory • require more communications overhead and, potentially, more controller memory than compound data types • can only change a tag's data type when programming offline • the root tag is listed alphabetically in the Tag Editor and Data Monitor, but the structure members are listed in the order in which they were defined in the structure

compound data type (array, structure)

Benefits:	Considerations:
• allow for specific names and user-defined organization • consolidates information in controller memory • optimizes communications time and memory impact • arrays can be dynamically indexed • can create new arrays when programming online • alias support for user-defined structures, members of an array, and bits of a member	• 2 Mbyte data limit per user-defined structure or array • user-defined structures are padded to enforce 32-bit data alignment • alias tags cannot point to the root tag of an array • Tag Editor and Data Monitor filtering limited • can only create or change a user-defined structure when programming offline • can only change an array when programming offline

The Logix CPU reads and manipulates 32-bit data values. All data starts at 32-bit offsets, so the minimum memory allocation for a tag is 4 bytes. When you create a standalone tag that stores data that is less than 4 bytes, the controller allocates 4 bytes, but the data only fills the part it needs.

Data type	Bits			
	31 16	15 8	7 1	0
BOOL	not used			0 or 1
SINT	not used		-128 to +127	
INT	not used	-32,768 to +32767		
DINT	-2,147,483,648 to +2,147,483,647			
REAL	$-3.40282347E^{38}$ to $-1.17549435E^{-38}$ (negative values) 0 $1.17549435E^{-38}$ to $3.40282347E^{38}$ (positive values)			

To manipulate SINT or INT data, the controller converts the values to DINT values, performs the programmed manipulation, and then returns the result to a SINT or INT value. This requires additional memory and execution time when compared to using DINT values for the same operation.

Guidelines for Data Types

Use DINT data types whenever possible

The Logix5000 controllers perform DINT (32 bit) and REAL (32 bit) math operations. DINT data types use less memory and execute faster than other data types. Use:

- DINT for most numeric values and array indexes
- REAL for manipulating floating-point, analog values
- SINT (8 bit) and INT (16 bit) primarily in user-defined structures or when communicating with an external device that does not support DINT values

	SINT	INT	DINT	REAL
memory reserved for a stand-alone tag	4 bytes	4 bytes	4 bytes	4 bytes
memory reserved for data in a user-defined structure	1 byte (8-bit aligned)	2 bytes (16-bit aligned)	4 bytes (32-bit aligned)	4 bytes (32-bit aligned)
memory used to access a tag in an ADD instruction	236 bytes	260 bytes	28 bytes	44 bytes
execution time on a 1756-L63 controller required to perform an ADD instruction	3.31 μsec	3.49 μsec	0.26 μsec	1.45 μsec

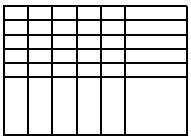
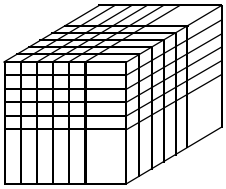
Group BOOL values into arrays

When using BOOL values, group them into DINT arrays to best use controller memory and to make the bits accessible via FBC or DDT instructions.

Arrays

An array allocates a contiguous block of memory to store a specific data type as a table of values.

- Tags support arrays in one, two, or three dimensions.
- User-defined structures can contain a single-dimension array as a member of the structure.

This array:	Stores data like:	For example:				
one dimension		Tag name:	Type	Dimension 0	Dimension 1	Dimension 2
		one_d_array	DINT[7]	7	--	--
		total number of elements = 7 valid subscript range DINT[x] where x=0–6				
two dimension		Tag name:	Type	Dimension 0	Dimension 1	Dimension 2
		two_d_array	DINT[4,5]	4	5	--
		total number of elements = 4 * 5 = 20 valid subscript range DINT[x,y] where x=0–3; y=0–4				
three dimension		Tag name:	Type	Dimension 0	Dimension 1	Dimension 2
		three_d_array	DINT[2,3,4]	2	3	4
		total number of elements = 2 * 3 * 4 = 24 valid subscript range DINT[x,y,z] where x=0–1; y=0–2, z=0–3				

The data type you select for an array determines how the contiguous block of memory gets used.

BOOL[96] = 12 bytes

BOOL arrays use 32-bit increments of memory

31	30	29	27	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
63	62	61	60	59	58	57	56	55	54	53	52	51	50	49	48	47	46	45	44	43	42	41	40	39	38	37	36	35	34	33	32
95	94	93	92	91	90	89	88	87	86	85	84	83	82	81	80	79	78	77	76	75	74	73	72	71	70	69	68	67	66	65	64

SINT[10] = 12 bytes of memory (2 bytes unused)

SINT arrays are padded to use any left over bytes

3	2	1	0
7	6	5	4
Unused	Unused	9	8

INT[5] = 12 bytes of memory (2 bytes unused)

INT arrays are padded to use any left over bytes

1	0
3	2
Unused	4

DINT[3] = 12 bytes and REAL[3] = 12 bytes

DINT and REAL arrays use 4-byte increments of memory

0
1
2

Guidelines for Arrays

You can create arrays of most data types, except for AXIS, MOTION_GROUP, and MESSAGE data types

A subscript identifies an individual element within the array. A subscript starts at 0 and extends to the number of elements minus 1 (zero based).

- Single-dimension arrays take less memory and execute faster than 2-dimension or 3-dimension arrays.
- Direct references to array elements execute faster than indexed references
- An array can be as large as 2 Mbytes
- If you create an array of structures, the memory for each element is allocated based on the structure definition

Type of Array	Benefits:	Considerations:
Single (1) dimension	• better support by native file instructions • fully supported in user-defined structures and arrays • smallest impact (execution time and memory) for indexed references • can create new arrays when programming online	• multiple arrays cannot be indirectly referenced like in PLC or SLC processors (i.e., N[N7:0]:5) • BOOL arrays not directly supported by file instructions • can only be changed when programming offline
Double (2) dimension and Triple (3) dimension	• can provide a more accurate data representation for a physical system • can emulate PLC file/word indirection with a 2 dimension array • can create new arrays when programming online	• larger impact (execution time and memory) for indexed references • file manipulation requires extra code in addition to file instructions • can only be changed when programming offline

Nest arrays

The file instructions offer limited support for arrays. To work with array data, create a user-defined structure with one array as a member of the structure. Then create an array tag using the user-defined structure as its data type.

Select the data type of the array based on the data, as well as the instructions that manipulate that data

While SINT and INT arrays can compact more values into a given memory area, they require additional memory and execution time for each instruction that references the array.

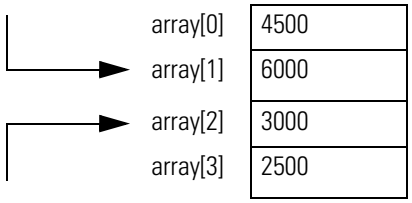
Limit arrays to 2 Mbytes of data

The maximum array size is 2 Mbytes. The software displays a warning if you try to create an array that is too large. The software also displays a warning if an array is 1.5-2 Mbytes in size, even though these sizes are valid.

Indirect Addressing of Arrays

If you want an instruction to access different elements in an array, use a tag in the subscript of the array (an indirect address). By changing the value of the tag, you change the element of the array that your logic references.

When *index* equals 1, *array[index]* points here.



When *index* equals 2, *array[index]* points here.

Directly referencing an element in an array (such as MyArray[20]), uses less memory and executes faster than an indirect reference (MyArray[MyIndex]). You can also indirectly address bits in a tag (MyDint.[Index]).

If you use indirect addresses, use DINT tags because other data types require conversion and execute slower. For each indexed access to data, the controller recalculates the array index. If you access a specific array element multiple times, copy the data out of the array into a fixed tag and use that tag in subsequent logic.

You can also use an expression to specify the index value. For example: MyArray[10 + MyIndex].

- An expression uses operators to calculate a value.
- The controller computes the result of the expression and uses it as the index.
- Valid operators include:

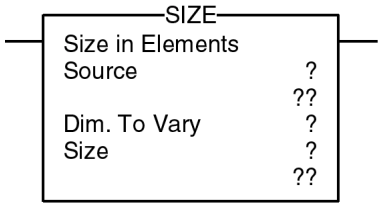
Operator:	Description:	Optimal:
+	add	DINT, REAL
-	subtract/negate	DINT, REAL
*	multiply	DINT, REAL
/	divide	DINT, REAL
**	exponent (x to y)	DINT, REAL
ABS	absolute value	DINT, REAL
ACS	arc cosine	REAL
AND	bitwise AND	DINT
ASN	arc sine	REAL
ATN	arc tangent	REAL
COS	cosine	REAL
DEG	radians to degrees	DINT, REAL
FRD	BCD to integer	DINT

Operator:	Description:	Optimal:
LN	natural log	REAL
LOG	log base 10	REAL
MOD	modulo-divide	DINT, REAL
NOT	bitwise complement	DINT
OR	bitwise OR	DINT
RAD	degrees to radians	DINT, REAL
SIN	sine	REAL
SQR	square root	DINT, REAL
TAN	tangent	REAL
TOD	integer to BCD	DINT
TRN	truncate	DINT, REAL
XOR	bitwise exclusive OR	DINT

Guidelines for Array Indexes

Use the SIZE instruction to determine the number of elements in an array

By determining the number of elements in an array at run time, you can write reusable code that adjusts itself to meet each instance where it is used.



Use immediate values to reference array elements

Immediate value references to array elements are quicker to process and execute faster than indexed references.

Use DINT tags for array indexes

DINT tags execute the fastest. SINT, INT, and REAL tags require conversion code that can add additional scan time to an operation.

Avoid using array elements as indexes

The Logix5000 controller does not directly support the use of an array element as the index to look up a value in another array. To work around this, you can create an alias to the element and then use this as the index. Or copy the element to a base tag and use that base tag as the index.

Prescan of an Array Index

During prescan, the controller resets state based on instructions such as outputs and timers. If you use calculated array indexes based on program execution, an "Indexed address out of range" error occurs because the program has not executed and the index was not initialized. You can use a fault handler routine to address this:

- Place an unconditional rung with an OTE instruction referencing an internal bit in the first program of the first task. During prescan, the prescan bit will be turned off. During normal scan, the prescan bit will be on at all times.
- "Indexed address out of range" error occurs and the prescan bit is off, reset the error and continue.

See the *Logix5000 Controllers Common Procedures Programming Manua*l, publication 1756-PM001 for information and sample code to handle faults.

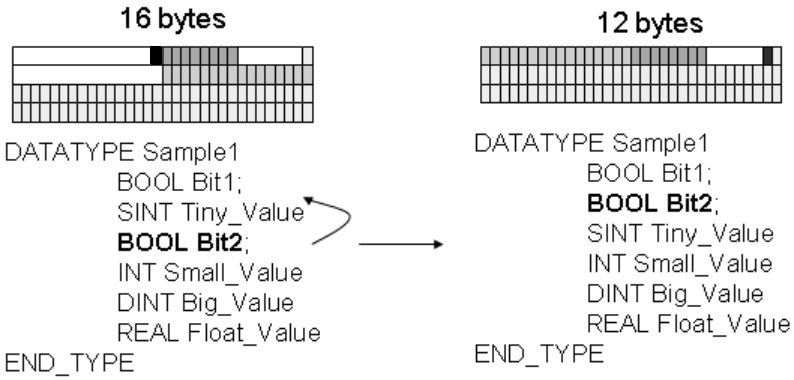
IMPORTANT This prescan condition no longer exists in controllers with firmware revision 13 and greater. You do not need to program a fault handler routine to handle indexed address out-of-range errors.

Guidelines for User-Defined Structures

User-defined structures let you combine multiple data types into one structure. All the elements in a structure align along 8-bit boundaries.

Group members of the same data type within a structure

When you use the BOOL, SINT, or INT data types, place members that use the same data type in sequence:



A Logix5000 controller aligns every data type along an 8-bit boundary for SINTs, a 16-bit boundary for INTS, or a 32-bit boundary for DINTs and REALs. BOOLs also align on 8-bit boundaries, but if they are placed adjacent to each other in a user-defined structure, they are mapped so that they share the same byte.

Arrays within structures can only be 1-dimension

If you include an array as a member, limit the array to a single dimension. Multi-dimension arrays are not permitted in a user-defined structure.

I/O data used in structure must be copied into the members

If you include members that represent I/O devices, you must use logic to copy the data into the members of the structure from the corresponding I/O tags.

Limit user-defined structures to 500 members

Logix5000 controllers limit user-defined structures to 500 members. If you need more, consider nesting structures within the main structure.

Limit the size of user-defined structures if they are to be communicated

Produced and consumed tags are limited to 500 bytes over the backplane and 480 bytes if over a network.

RSLinx can optimize user-defined structures that are less than 480 bytes.

Selecting a Data Type for Bit Tags

Bits in a Logix5000 controller can exist as: BOOL tags, bits in a BOOL array, bits in elements of a SINT, INT, DINT array, members of a user-defined structure, or as bits in a SINT, INT, DINT member of a user-defined structure.

BOOL tag

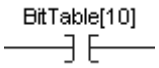
MyBit:BOOL

Each tag accesses a specific bit. Each tag uses 4 bytes.

Benefits:	Considerations:
• each bit has a specific tag	• requires extra bandwidth to communication • uses more memory; 32-bits for each tag • cannot use FBC/DDT bit file instructions

BOOL array

BitTable:BOOL[32]

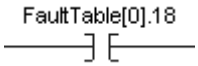


A BOOL array combines multiple bits into adjacent words (32-bit words).

Benefits:	Considerations:
• consolidates multiple bits into a single word • better use of memory • can address all bits in an array using indirect addressing	• BOOL data type only supported by bit instructions • cannot use file instructions, copy instructions, or DDT/FBC instructions

DINT array

FaultTable:DINT[3]



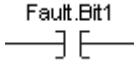
A DINT combines multiple bits into adjacent words.

Benefits:	Considerations:
• consolidates multiple bits into a single word • file instructions, copy instructions, and DDT/FBC instructions support DINT arrays • lets you access the bits by element (word) and bit number	• requires extra planning to indirectly address bits • difficult to address bits in the array using indirect addressing

user-defined structure

BitStructure
  Bit1:BOOL
  Bit2:BOOL

Fault:BitStructure



A user-defined structure combines multiple bits into adjacent, individually-named words.

Benefits:	Considerations:
• object based • consolidates multiple bits into a single word	• structures are not directly supported by 3rd party MMI/EOI products (RSView does support 32-bit tags and structures) • cannot use FBC/DDT bit file instructions

Serial Bit Addressing

The BOOL “B” data table in the PLC-5 and SLC 500 processors supports two addressing modes that can address the same bit:

Addressing mode:	Description:
serial bit In PLC-5 or SLC software, this addressing mode is represented as “/Bit”	Serial bit addressing provides the ability to reference all bits as a contiguous list (array) of bits. For example, if you wanted to reference the 3rd bit in the 2nd word of a “B” file, you specify B3/18. This method similar to a BOOL array in a Logix5000 controller where you would specify FaultBit[18].
word bit In PLC-5 or SLC software, this addressing mode is represented as “Word/Bit”	Word bit addressing identifies a bit within a specific word. For example, B3:1/2 is the same as B3/18 from the serial bit example. This method is similar to accessing the bits of a SINT, INT, DINT array in a Logix5000 controller where you would specify FaultTable[1].2.

The Logix5000 controller supports both of these addressing modes, but you cannot use both to reference bits in the same array due to conformance with the IEC 61131-3 standard. Choose the method that best meets your application needs. You can copy data between arrays using both methods.

You can also use an expression to indirectly reference a bit in a DINT array using a serialized bit number. For example:

```
Tag
       MyBits : DINT[10]
       BitRef : DINT
EndTag

MOV(34, BitRef)
XIC(MyBits[BitRef / 32].[BitRef AND 31])
```

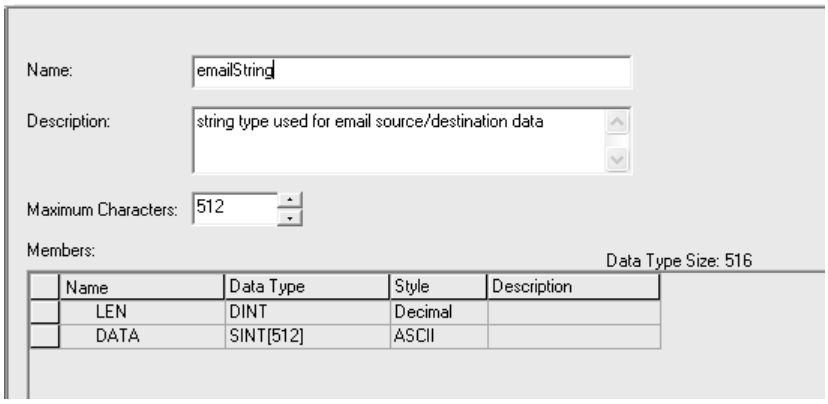
where:

This expression:	Calculates the:
`[BitRef / 32]`	element in the DINT array
Note: If the tag MyBits is an INT or SINT, the divisor would be 16 or 8, respectively.	
`[BitRef AND 31]`	bit within the element
Note: If the tag MyBits is an INT or SINT, the mask value would be 15 or 7, respectively.	

The Diagnostic Detect (DDT) and File Bit Compare (FBC) instructions provide a bit number as a result of their operation. These instructions are limited to DINT arrays so you can use them to locate the bit number returned from the example above.

Guidelines for String Data Types

String data types are structures that hold ASCII characters. The first member of the structure defines the length of the string; the second member is an array that holds the actual ASCII characters.



You can create a string data type that is longer or shorter than the default string data type

The default string data type can contain as many as 82 characters, but you can create custom-length string data types to hold as many characters as needed.

Only some instructions support string data types

These comparison instructions support string tags: EQU, NEQ, GRT, GEG, LES, LEQ, CMP.

These serial port instructions support string tags: ARD, ARL, AWA, AWT.

These string-handling instructions support string tags: STOD, DTOS, STOR, RTOS, CONCAT, MID, FIND, DELETE, INSERT, UPPER, LOWER, SIZE.

These file instructions support string arrays: FAL, FFL, FFU, LFL, LFU, COP, CPS, FSC.

Use the SIZE instruction to determine the number of characters in a string

By determining the number of characters in a string at run time, you can write reusable code that adjusts itself to meet each instance where it is used.

Using the DTOS, RTOS, and CONCAT instructions, you can embed tag values within a string

The SLC 500 processor supports the ability to embed a data table reference address within a string (inline indirection). The SLC 500 AWA and AWT instructions can then look up the data value and place an ASCII representation into the outgoing string. The Logix5000 controller does not directly support this ability. Use the DTOS or RTOS instructions to convert a value to a string and the CONCAT instruction to merge characters with another string.

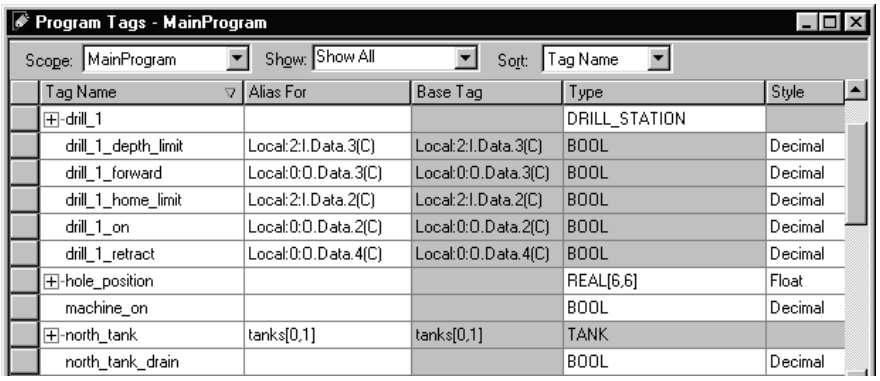
PLC-5/SLC 500 Access of Strings

The ASCII “A” data table in the PLC-5 and SLC 500 processors uses a string format that is similar to the Logix string data type. The main difference is that the LEN field (length) in a PLC-5/SLC 500 processor is a 16-bit, INT value whereas the LEN field in a Logix5000 controller is a 32-bit, DINT field. This difference can impact converted logic and data communications. The Logix5000 controller will convert the LEN field to the appropriate value and size when a PLC-5/SLC 500 message format is used to read or write a string.

Configuring Tags

A tag is a text-based name for an area of the controller's memory where data is stored. Tags are the basic mechanism for allocating memory, referencing data from logic, and monitoring data.

If you want the tag to:	Then choose this type:
store a value for use by logic within the project	Base
use a different name for an existing tag's data (can help simplify long, pre-determined tag names, such as for I/O data or user-defined structures)	Alias
send (broadcast) data to another controller	Produced
receive data from another controller	Consumed



Tag Name	Alias For	Base Tag	Type	Style
+-drill_1			DRILL_STATION	
drill_1_depth_limit	Local:2:I.Data.3(C)	Local:2:I.Data.3(C)	BOOL	Decimal
drill_1_forward	Local:0:O.Data.3(C)	Local:0:O.Data.3(C)	BOOL	Decimal
drill_1_home_limit	Local:2:I.Data.2(C)	Local:2:I.Data.2(C)	BOOL	Decimal
drill_1_on	Local:0:O.Data.2(C)	Local:0:O.Data.2(C)	BOOL	Decimal
drill_1_retract	Local:0:O.Data.4(C)	Local:0:O.Data.4(C)	BOOL	Decimal
+-hole_position			REAL[6,6]	Float
machine_on			BOOL	Decimal
+-north_tank	tanks[0,1]	tanks[0,1]	TANK	
north_tank_drain			BOOL	Decimal

For more information on I/O tags, see Chapter 6 "*Communicating with I/O.*"

Guidelines for Base Tags

Create stand-alone atomic tags

The controller supports pre-defined, stand-alone tags.

- Atomic tags are listed directly in the Tag Editor and Data Monitor and can be easily located by browsing the alphabetical list.
- Atomic tags can be created on-line, but the data type can only be modified off-line.
- Using only atomic tags can impact HMI communications performance as more information must be passed and acted on.

Create user-defined structures

User-defined structures (data types) let you organize your data to match your machine or process.

- One tag contains all the data related to a specific aspect of your system. This keeps related data together and easy to locate, regardless of its data type.
- Each piece of data (member) gets a descriptive name.
- You can use the structure to create multiple tags with the same data layout.
- User-defined structure can only be modified off-line
- RSLinx optimizes user-defined structures more than stand-alone tags.

Use arrays like files to quickly create a group of similar tags

An array creates multiple instances of a data type under a common tag name.

- Arrays let you organize a block of tags that use the same data type and perform a similar function.
- You organize the data in 1, 2, or 3 dimensions to match what the data represents.
- Arrays can only be modified off-line.
- RSLinx optimizes array data types more than stand-alone tags.

Minimize the use of BOOL arrays. Many array instructions *do not* operate on BOOL arrays. This makes it more difficult to initialize and clear an array of BOOL data.

Take advantage of program-scoped tags

If you want multiple tags with the same name, define each tag at the program scope (program tags) for a different program. This lets you re-use both logic and tag names in multiple programs.

Avoid using the same name for both a controller tag and a program tag. Within a program, you cannot reference a controller tag if a tag of the same name exists as a program tag for that program.

Use mixed case and the underscore characters

Although tags are not case sensitive (upper case *A* is the same as lower case *a*), mixed case is easier to read. For example, "Tank_1" can be easier to read than "tank1."

Consider alphabetical order

RSLogix 5000 software displays tags of the same scope in alphabetical order. To make it easier to monitor related tags, use similar starting characters for tags that you want to keep together. For example, consider using "Tank_North" and "Tank_South" rather than North_Tank" and "South_Tank."

Use leading zeroes (0) when numbers are part of tag names

RSLogix 5000 software uses a simple sort to alphabetize tag names in the Tag Editor and Data Monitor. This means if you have Tag1, Tag2, Tag11, and Tag12, the software displays them in order as Tag1, Tag11, Tag12, and then Tag2. If you want to keep them in numerical order, name them Tag01, Tag02, Tag11, and Tag12.

Creating Alias Tags

An alias tag lets you create one tag that represents another tag.

- both tags share the same value as defined by the base tag
- when the value of a base tag changes, all references (aliases) to the base tag reflect the change

An alias tag references a base tag

When assigning aliases, avoid

- nesting aliases (you cannot have an alias of an alias)
- using multiple aliases to the same tag

On upload, the software decompiles the program and uses the physical memory addresses to determine which tags are referenced in the code. All references to a base tag reverts to an alias if one exists. If multiple aliases point to the same tag, RSLogix 5000 software uses the first alias tag (alphabetically) that it finds.

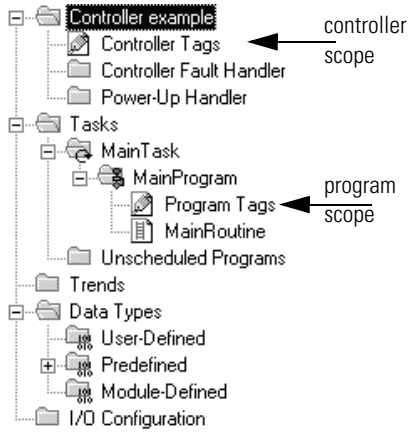
Alias tags do not affect controller execution

During download, the program is compiled into machine executable code and physical memory addresses. While the existence of an alias requires controller memory to store the name, the program performs the same operation for a reference with an alias or its associated base tag.

Accessing alias tags from RSLinx software

Because an alias tag appears as a stand-alone tag to RSLinx software, an alias tag that references a compound array or structure might require additional communication time. When referencing tags from RSLinx software or other HMI, it might be fastest to reference base tags directly.

Guidelines for Data Scope



Data scope defines where you can access tags. When you create a tag, you assign scope as either controller or program. Controller-scoped tags are accessible by all programs. Program-scoped tags are only accessible by the code within a specific program.

If you want to:	Then assign this scope:
use a tag in more than one program in the same project	controller scope (controller tags)
use a tag in a message (MSG) instruction	
produce or consume data	
use motion tags	
communicate with a PanelView terminal	
reuse the same tag name multiple times for different parts or processes within a controller	program scope (program tags)
have multiple programmers working on logic and you want to merge logic into one project	

Isolate portions of a machine or different stations into separate programs and use program-scoped tags within each program. This:

- provides isolation between programs
- prevents tag name collisions
- improves the ability to reuse code

Notes:

Sharing Tag Data with Other Controllers (Produced and Consumed Tags)

Introduction

Logix5000 controllers support the ability to produce (broadcast) and consume (receive) system-shared tags.

For two controllers to share produced or consumed tags, both controllers must be attached to the same control network (such as a ControlNet or Ethernet/IP network). You cannot bridge produced and consumed tags over two networks.

Logix5000 controllers can produce and consume tags over these networks (as long as they support communications over these networks):

- the ControlLogix backplane
- a ControlNet network
- an EtherNet/IP network

If there are no other connections, the controller supports:

As a:	The controller supports:
producer	*(number of produced tags)* ≤ 127
consumer	*(number of consumed tags)* ≤ 250 (or controller maximum)

The total combined number of consumed and produced tags that a controller supports is:

(produced tags) + (consumed tags) + (other connections) ≤ 250 (*or controller maximum*)

IMPORTANT The actual number of produced and consumed tags that you can configure in project depends on the connection limits of the communication module through which you produce or consume the tags.

Guidelines for Creating Produced and Consumed Tags

You cannot bridge produced and consumed tags over different networks	For two controllers to share produced or consumed tags, both controllers must be attached to the same network. You can produce and consume tags over ControlNet or EtherNet/IP networks.
Create the tag at controller scope	You can only produce and consume (share) controller-scoped tags.
Limit the size of the tag to ≤ 500 bytes	If you transfer a tag with more than 500 bytes, create logic to transfer the data in packets. If you consume a tag over a ControlNet hop, the tag must be ≤ 480 bytes. This is a limitation of the ControlNet network, not the controller.
Combine data that goes to the same controller	If you are producing several tags for the same controller: • Group the data into one or more user-defined structures. This uses less connections than producing each tag separately. • Group the data according to similar update intervals. To conserve network bandwidth, use a greater RPI for less critical data.
Use one of these data types: • **DINT** • **REAL** • **array of DINTs or REALs** • **user-defined structure**	To share data types other than DINT or REAL, create a user-defined structure to contain the required data. Use the same data type for the produced tag and the corresponding consumed tag or tags.
Use a user-defined structure to produce or consume INT or SINT data	To produce or consume INT or SINT data, create a user-defined structure with INT or SINT members. The members can be individual INTs or SINTs or the members can be INT or SINT arrays. The resulting user-defined structure can then be produced or consumed.
The data type in the producer and the consumer must match	The data type for a produced or consumed tag must be the same in both the producer and the consumer.
Produce tags based on user-defined structures to non-Logix devices	The controller produces tags in 32-bit words. For devices that communicate in other word boundaries, such as 16-bit words, the resulting data in the target device can be misaligned. To help avoid misalignment, structure the produced data in a user-defined structure.
Use a CPS instruction to buffer produced and consumed data	Use the CPS instruction to copy the data to the outgoing tag on the producer side. Then use another CPS instruction to copy the data into a buffer tag on the consumer side. The CPS instructions provides data integrity for data structures greater than 32 bits.

Guidelines for Specifying an RPI Rate

When configuring produced and consumed tags, you specify an Requested Packet Interval (RPI) rate. The RPI value is the rate at which the controller attempts to communicate with the module.

Make sure the RPI is equal to or greater than the NUT	You use RSNetWorx for ControlNet software to select the network update time (NUT) and the software schedules the network connections. RSNetWorx cannot schedule a ControlNet network if a module and/or produced/consumed tag on the network has an RPI that is faster than the network update time.
The smallest (fastest) consumer RPI determines the RPI for the produced tag	If multiple consumers request the same tag, the smallest (fastest) request determines the rate at which the tag is produced for all the consumers.

Guidelines for Managing Connections for Produced and Consumed Tags

Minimize the use of produced and consumed tags	To reduce network traffic, minimize the size of produced and consumed tags. Also, minimize the use of produced and consumed tags to high-speed, deterministic data, such as interlocks.
Use arrays or user-defined structures	When sending multiple tags to the same controller, use an array or user-defined structure to consolidate the data. The byte limit of ≤ 500 bytes per produced and consumed tag still applies.
Configure the number of consumers accurately	Make sure the number of consumers configured for a produced tag is the actual number of controllers that will consume the tag. If you set the number higher than the actual number of controllers, you unnecessarily use up connections. The default is 2 consumers per produced tag.
Multiple produced/consumed connections are linked	If there are multiple produced and consumed connections between two controllers and one connection fails, all the produced and consumed connections fail. Consider combining all produced and consumed data into one structure or array so that you only need one connection between the controllers.

Configuring an Event Task Based on a Consumed Tag

An event task executes automatically based on a preconfigured event occurring. One such event can be the arrival of a consumed tag.

- Only one consumed tag can trigger a specific event task.
- Typically, use an IOT instruction in the producing controller to signal the production of new data.
- When a consumed tag triggers an event task, the event task waits for all the data to arrive before the event task executes.

For information on configuring an event task, see Chapter 2 "*Dividing Logic into Tasks, Programs, and Routines*."

Comparing Messages and Produced/Consumed Tags

Method:	Benefits:	Considerations:
Read/Write Message	• programatically initiated • communications and network resources only used when needed • support automatic fragmentation and reassembly of large data packets, up to as many as 32,767 elements • some connections can be cached to improve re-transmission time • Generic CIP message useful for third-party devices	• controller limited to 32 active messages active at the same time (limit of 16 in revision 11 and earlier) • delay may occur if resources are not available when needed • MSG instruction and processing impacts controller scan (system overhead timeslice) • data arrives asynchronous to program scan (use CPS instruction to reduce impact, no event task support) • fragmentation and reassembly limited to exchanges between Logix5000 controllers
Produced/Consumed Tag	• configured once and sent automatically based on Requested Packet Interval (RPI) • multiple consumers can simultaneously receive the same data from a single produced tag • can trigger an event task when consumed data arrives • ControlNet resources are reserved up front • does not impact the scan of the controller	• support limited to Logix5000 and PLC-5 controllers, and the 1784-KTCS I/O Linx and select third party devices • limited to 500 bytes over the backplane and 480 bytes over a network • must be scheduled when using ControlNet • data arrives asynchronous to program scan (use CPS instruction and event tasks to synchronize) • connection status must be obtained separately

Designing Networks

Introduction

NetLinx Open Network Architecture is the Rockwell Automation strategy of using open networking technology for seamless, top-floor to shop-floor integration. The networks in the NetLinx architecture — DeviceNet, ControlNet, and EtherNet/IP — share a universal set of communication services. These are the recommended networks for Logix control systems.

Select a Network

Comparison:	EtherNet/IP:	ControlNet:	DeviceNet:
Control I/O	better	BEST	low density
Configuration devices	BEST	BEST	BEST
Collect data	BEST	Better	good
Peer interlocking	better	BEST	good
Devices	better	better	BEST
Topologies	star requires switches	trunkline/dropline star with repeaters	trunkline/dropline
Capacity	many nodes	99 nodes	63 nodes
Performance	BEST	BEST	good

EtherNet/IP Network Topology

EtherNet/IP network:

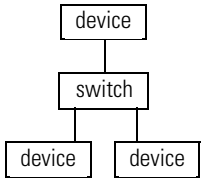
- EtherNet/IP supports messaging, produced/consumed tags, and distributed I/O
- EtherNet/IP supports half/full duplex 10 Mbps or 100 Mbps operation
- EtherNet/IP requires no network scheduling and no routing tables
- There are several methods available to configure EtherNet/IP network parameters for devices. Not all methods are available at all times. These methods are device and configuration dependent.
  - DHCP
  - Rockwell Automation BOOTP/DHCP utility
  - RSLinx software
  - RSLogix 5000 software
  - RSNetWorx for EtherNet/IP software

Application Ideas

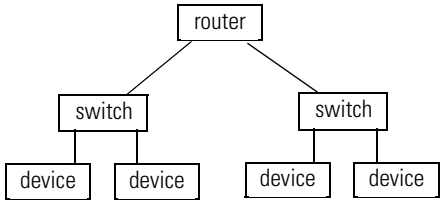
- connect many computers
- default gateway to business systems
- star topology best for few nodes and short distances

Topology:

example 1



example 2



Guidelines for EtherNet/IP

Make sure the switch has the required features

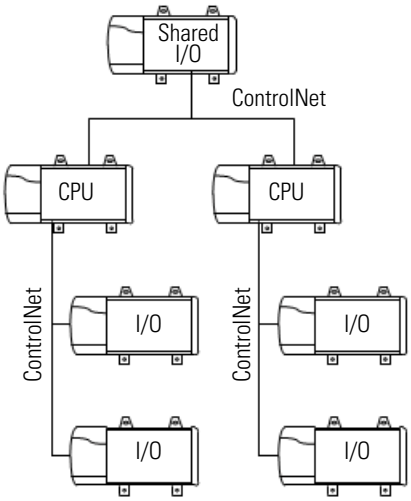
For EtherNet/IP control, you must use an industrial-grade switch that supports:

- full-duplex on all ports
- IGMP snooping
  - constrains multicast traffic to ports associated with a specific IP multicast group
  - most switches require a router for IGMP snooping to function
  - if you have a stand-alone network, make sure the switch supports IGMP snooping without a router present
- port mirroring
- VLAN (virtual local area network) to isolate traffic flow for different systems
- both autonegotiation and manual configuration of duplex and speed
- wire-speed switching fabric
- SNMP (Simple Network Management Protocol) to obtain statistical information about a device
- IEEE 802.1D spanning tree protocol to support redundant backbone connections for improved fault tolerances
- IEEE 802.1P frame prioritization
- IP address blocking to restrict traffic to a specific range
- auto-restore of switch configuration for replacement
- per port broadcast and multicast storm control
- port trunking for applications with multiple switches
- method for backing up configuration information

Consider using switches from Encompass partners

These Encompass partners have switches that meet the required features: Cisco, Hirschmann, and N-Tron.

ControlNet Network Topology

ControlNet network:	Topology:
• ControlNet allows both I/O and messaging on the same wire. • Multiple controllers and their respective I/O can also be placed on the same ControlNet wire. • When new I/O is added or an existing I/O module's communication structure is changed, you must use RSNetWorx for ControlNet software to reschedule the network. • If the network timing changes, every device with scheduled traffic on the network is affected. • To reduce the impact of changes, place each CPU and its respective I/O on isolated ControlNet networks. • Place shared I/O and produced/consumed tags on a common network available to each CPU that needs the information. **Application Ideas** • default Logix network • best replacement for Universal Remote I/O • backbone to multiple distributed DeviceNet networks • peer interlocking network • common devices include: Logix5000 controllers, PanelView terminals, I/O modules, and drives	

Guidelines for ControlNet

Use the installation publications when installing a ControlNet network

Use these publications when installing a ControlNet network:

- *ControlNet Coax Media Planning and Installation Guide*, publication CNET-IN002
- *ControlNet Fiber Media Planning and Installation Guide*, publication CNET-IN001

Limit the number of nodes per ControlNet network to 40

ControlNet was designed with a limit of 99 nodes per network, but this number of nodes decreases network performance. A maximum of 40 nodes per network results in better performance and leaves bandwidth for other communications.

Adjust the default RSNetWorx settings

Change these settings in the RSNetWorx for ControlNet software:

- UMAX (highest unscheduled node on the network)
  - default is 99
  - the network takes the time to process the total number of nodes specified in this setting, even if there are not that many devices on the network
  - change to a reasonable level to accommodate the active devices on the network and any additional devices that might be connected
- SMAX (highest scheduled node on the network)
  - default is 1
  - this must be changed for all systems
  - set SMAX < UMAX

Design for at least 400 Kbytes of available, unscheduled network bandwidth

Leaving too little memory for unscheduled network bandwidth results in poor message throughput and slower workstation reponse.

Place DeviceNet (DNB) and serial (MVI) communication modules in the local chassis

DeviceNet (DNB) and serial (MVI) communication modules have multiple, 500-byte data packets that will impact scheduled bandwidth. Placing these modules in the same chassis as the controller avoids this data being scheduled over the ControlNet network

If you must place these communication devices in remote chassis, configure the input and output sizes to match the data configured in RSNetWorx for DeviceNet software. This reduces the amount of data that must be transmitted.

Limit 1756-CNB, -CNBR connections

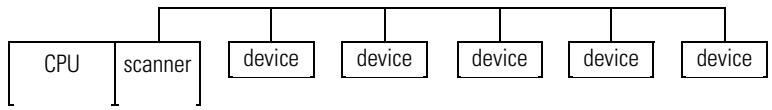
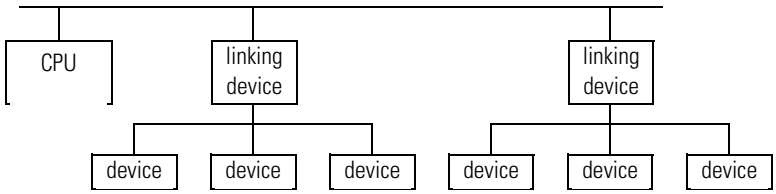
For best performance, limit the 1756-CNB, -CNBR to 40-48 connections. Add additional modules in the same chassis if you need more connections. Adding more modules and splitting connections among the modules can improve system performance.

If the chassis that contains the CNB module also contains multiple digital I/O modules, configure the CNB module's communication format for "Rack Optimization." Otherwise, use "None." See the examples on page 6-5.

If you change network settings, resave each controller's project

Any time you use RSNetWorx software and you save or merge your edits, attach to each controller in the system with their respective RSLogix 5000 project file and perform a save. This copies the ControlNet settings into the offline, database file and ensures that future downloads of the controller permit it to go online without having to run RSNetWorx software.

DeviceNet Network Topology

DeviceNet network:	Topology:
• You need a DeviceNet scanner to connect the controller to DeviceNet devices • You must use RSNetWorx for DeviceNet to configure devices and create the scanlist for the scanner. • You can configure the network baud rate as 125K bit/s (default and a good starting point), 250K bit/s, or 500K bit/s. • If each device on the network (except the scanner) sends ≤ 4 bytes of input data and receives ≤ 4 bytes of output data, you can use the AutoScan feature on the scanner to configure the network. **Application Ideas** • distributed devices • drives network • diagnostic information	**single network**  **several smaller distributed networks (subnets)** 

Guidelines for DeviceNet

Use the installation publications when installing a DeviceNet network

Use this publication when installing a DeviceNet network:

- *DeviceNet Cable System Manual*, publication DN-UM072

Place DeviceNet (DNB) communication modules in the local chassis

Placing DNB modules in the local chassis maximizes performance, especially in ControlLogix systems.

Size the input and output image for the DNB modules to the actual devices that are connected plus 20% for future growth. If you have to place DNB modules in remote chassis, sizing the input and output images is critical for best performance.

Verify the total network data does not exceed the maximum DNB data table size

A DNB supports:

- 124, 32-bit input words
- 123 32-bit output words
- 32, 32-bit status words

You can use RSNetWorx for DeviceNet software offline to estimate network data. Use a second DNB if there is more network data than one module can support.

Set up slaves first

Configure a device's parameters before adding that device to the scanlist. You cannot change the configuration of many devices once they are already in the scanlist.

If you configure the scanner first, there is a chance that the scanner configuration will not match the current configuration for a device. If the configuration does not match, the device will not show up when you browse the network.

Leave node address 63 open to add nodes

Devices default to node 63 out-of-the-box. Leave node address 63 unused so you can add a new devices to the network. Then change the address of the new device.

Leave node address 62 open to connect a computer

Always leave at least one open node number to let a computer be attached to the network if needed for troubleshooting, configuration, etc.

Don't forget to set the scanner run bit

For the scanner to be in Run mode, the controller must be in Run mode and the logic in the controller must set the scanner's run bit.

Make sure you have the most current EDS files for your devices

RSNetWorx for DeviceNet software uses EDS file to recognize devices. If the software is not properly recognizing a device, you are missing the correct EDS file(s). For some devices, you can create an EDS file by uploading information from the device. Or you can get EDS files from: http://www.ab.com/networks/eds.

Notes:

Communicating with I/O

Introduction

In Logix5000 controllers, I/O values update at a period (Requested Packet Interval, RPI) that you configure via Module Property dialog in the I/O configuration folder of the project. The values update asynchronously to the execution of logic.

The module sends input values to the controller at the specified RPI. Because this transfer is asynchronous to the execution of logic, an I/O value in the controller can change in the middle of a scan.

Buffering I/O Data

If you reference an I/O tag multiple times and the application could be impacted if the value changes during a program scan, you must copy the I/O value into a buffer tag prior to the first reference of that tag in your code. In your code, reference the buffer tag rather than the I/O tag.

Use the synchronous copy (CPS) instruction to buffer I/O data. While the CPS instruction copies data, no I/O updates or other tasks can change the data. Tasks that attempt to interrupt a CPS instruction are delayed until the instruction is done. Buffer I/O data to:

- prevent an input or output value from changing during the execution of a program. (I/O updates asynchronous to the execution of logic.)
- copy an input or output tag to a member of a structure or element of an array.
- prevent produced or consumed data from changing during the execution of a program.
- ensure all produced and consumed data arrives or is sent as a group (not mixed from multiple transfers)
- only use the CPS instruction if the I/O data that you want to buffer is greater than 32 bits (or 4 bytes) in size

If you have a user-defined structure with members that represent I/O devices, you must use logic to copy the data into the members of the structure from the corresponding I/O tags

Guidelines for Specifying an RPI Rate for I/O Modules

When adding I/O modules to a controller project, you specify a Requested Packet Interval (RPI) rate. Depending on the controller platform, you can select an RPI rate per module (ControlLogix) or an RPI rate per controller (CompactLogix and FlexLogix).

The RPI value is the rate at which the controller attempts to communicate with the module.

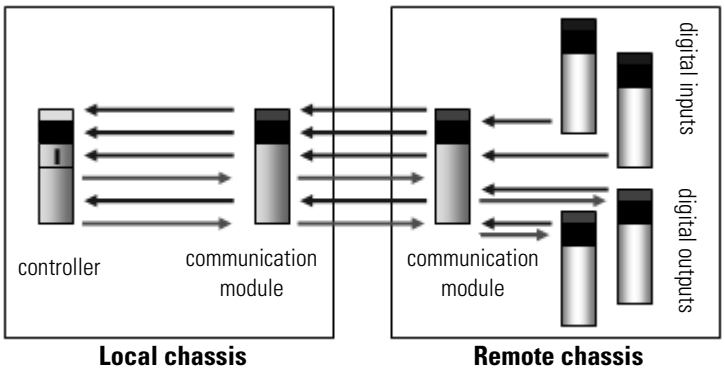
Specify an RPI at 50% of the rate you actually need	Setting the RPI faster (specifying a smaller number) than what your application needs wastes network resources, such as ControlNet schedule bandwidth, network processing time, and CPU processing time. For example, if you need information every 80 msec, set the RPI at 40 msec. The data is asynchronous to the controller scan, so you sample data twice as often (but no faster) than you need it to make sure you have the most current data.
Group devices with similar performance needs onto the same module	By grouping devices with similar performance needs on the same module, you consolidate data transmission to one module rather than multiple modules. This conserves network bandwidth.
Set the ControlNet network update time (NUT) equal to or less than the fastest RPI	When configuring a ControlNet network, set the network update time (NUT) equal to or less than the fastest RPI of the I/O modules and produced/consumed tags in the system. For example, if your fastest RPI is 10 msec, set the NUT to 5 msec for more flexibility in scheduling the network.
The RPI should be an even multiple of the NUT	Set the RPI to a binary multiple of the NUT. For example, if the NUT is 10 msec, select an RPI such as 10, 20, 40, 80, 160, etc. msec.

Communication Formats for I/O Modules

The communication format determines whether the controller connects to the I/O module via a direct or a rack-optimized connection. The communication format also determines the type and quantity of information that the module will provide or use.

direct connection

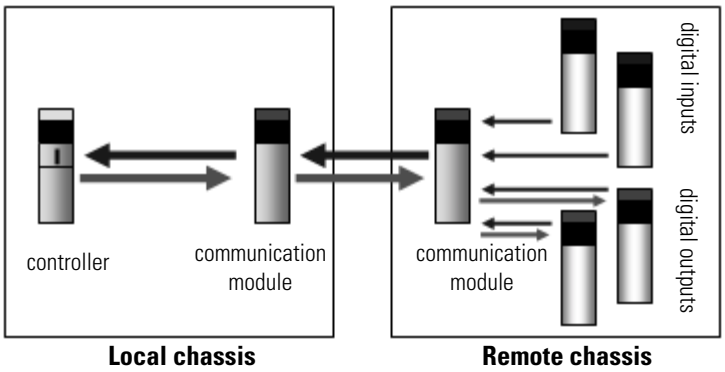
Each module passes its data to/from the controller individually. Communication modules bridge data across networks.



Benefits:	Considerations:
• each module can determine its own rate (RPI) • more data can be sent per module, such as diagnostic and analog data • supports event task communications	• requires additional connections and network resources • this is the only method supported in the local chassis • I/O data presented as individual tags

rack-optimized connection

The communications module in a remote chassis consolidates data from multiple modules into a single packet and transmits that packet as a single connection to the controller.



.

Benefits:	Considerations:
• one connection can service a full chassis of digital modules • reduces network resources and loading	• all modules are sent at the same rate • unused slots are still communicated • still need a direct connection for analog and diagnostic data • limited to remote chassis • I/O data presented as arrays with alias tags for each module

The rack-optimized format limits data to a single 32-bit input word per module in a chassis. If you place a diagnostic module in a chassis, the rack-optimized format eliminates the value that the diagnostic module offers. In this case, it's better to use a direct connection so that all of the module's diagnostic information is passed to the controller.

Guidelines for Managing I/O Connections

The type of I/O module can determine the type of connection

Analog modules always use direct connections, except for 1771 analog modules which use connected messaging.

Digital modules can use direct or rack-optimized connections. Communication formats that include "optimization" in the title are rack-optimized connections; all other connection options are direct connections.

Select the communication format for a remote adapter based on the remote I/O modules

For a remote adapter:

Select:	If:
None	the remote chassis contains only analog modules, diagnostic digital modules, fused output modules, or communication modules
Rack-Optimized	the remote chassis only contains standard, digital input and output modules (no diagnostic modules or fused output modules)
Listen Only Rack-Optimized	you want to receive I/O module and chassis slot information from a rack-optimized remote chassis owned by another controller

Use rack-optimized connections to conserve connection use

If you are trying to limit the number of controller and network connections, rack-optimized connections can help.

Guidelines for Managing I/O Connections (continued)

In some cases, all direct connections work best

For a remote adapter module configured for rack-optimized connections, there is always data sent for each slot in the chassis, even if a slot is empty or contains a direct connection module. There are 12 bytes of data transferred for rack-optimized overhead between the controller and the remote adapter module. In addition, the remote adapter module sends 8 bytes per slot to the controller; the controller sends 4 bytes per slot to the remote adapter.

For a small number of digital modules in a large chassis, it might be better to use direct connections because transferring the full chassis information might require more system bandwidth than direct connections to a few modules.

For example:

Example:	Description:
Remote 17-slot chassis Slot 0: 1756-CNBR/D Slots 1-15: analog modules Slot 16: standard digital module	**Option 1:** Select Rack Optimization for remote adapter's communication format. This example uses 16 controller connections (15 for analog modules and 1 for the rack-optimized connection). This example also transfers: • 12 bytes for rack-optimized overhead • 12 bytes for the digital module • 12 bytes for each of the 15 analog modules, for a total of 180 bytes **Option 2:** Select None for the remote adapter's communication format. This example also uses 16 controller connections (1 direct connection to each I/O module). There is no rack-optimized overhead data to transfer. **Recommendation:** Option 2 is recommended because it avoids unnecessary network traffic, and thus improves network performance.
Remote 17-slot chassis Slot 0: 1756-CNBR/D Slots 1-8: analog modules Slots 9-16: digital modules	**Option 1:** Select Rack Optimization for the remote adapter's communication format. This example uses 9 controller connections (8 for analog modules and 1 for the rack-optimized connection). This example also transfers: • 12 bytes for rack-optimized overhead • 12 bytes for each of the 8 digital modules, for a total of bytes 96 bytes • 12 bytes for each of the 8 analog modules, for a total of 96 bytes **Option 2:** Select None for remote adapter's communication format. This example uses 16 controller connections (1 direct connection to each I/O module). There is no rack-optimized overhead data to transfer. **Recommendation:** The best option for this example depends on the type of digital I/O modules in the system and other controller connections. If the total system has many analog modules, diagnostic modules, fused output modules, or produced/consumed tags, select Option 1 to conserve controller connections. If there are plenty of controller connections available, select Option 2 to reduce unnecessary network traffic.

Creating Tags for I/O Data

Each I/O tag is automatically created when you configure the I/O module through the programming software. Each tag name follows this format:

Location:SlotNumber:Type.MemberName.SubMemberName.Bit

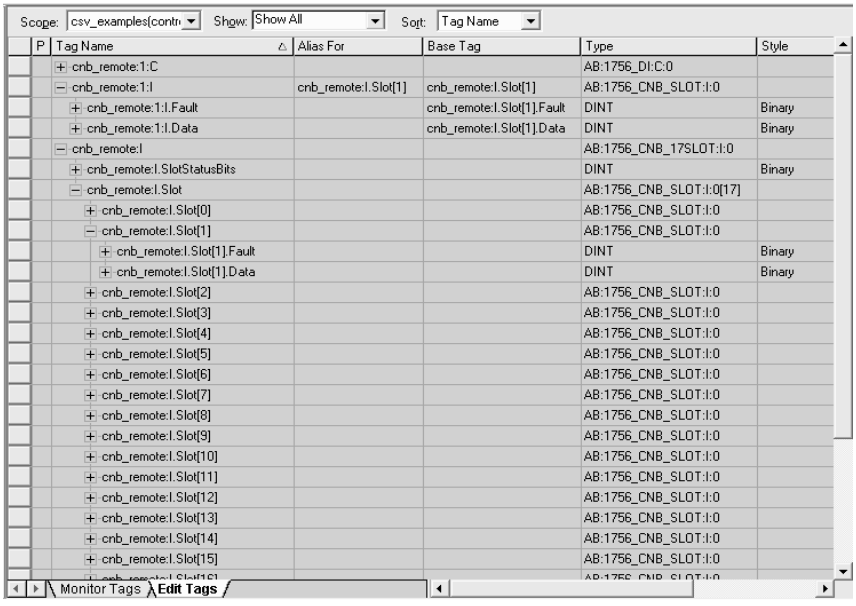
This address variable:	Is:
Location	Identifies network location LOCAL = local chassis or DIN rail ADAPTER_NAME = identifies remote adapter or bridge
SlotNumber	Slot number of I/O module in its chassis
Type	Type of data: I = input C = configuration O = output S = status
MemberName	Specific data from the I/O module, such as Data and Fault; depends on the module
SubMemberName	Specific data related to a MemberName.
Bit (optional)	Specific point on the I/O module; depends on the size of the I/O module (0-31 for a 32-point module)

If you configure a rack-optimized connection, the software creates a rack-object tag for the remote communication module. You can reference the rack-optimized I/O module individually, or by its element within the rack-object tag.

For example, a remote ControlNet communication module (remote_cnb) has an I/O module in slot 1.

This is the individual tag created for the I/O module in remote slot 1.

This is the entry in the rack-object tag for the remote communication module that identifies the I/O module in remote slot 1.



Controller Ownership

When you choose a communication format, you have to choose whether to establish an owner or listen-only relationship with the module.

owner	The owner controller writes configuration data and can establish a connection to the module.
listen-only	A controller using a listen-only connection only monitors the module. It does not write configuration data and can only maintain a connection to the I/O module when the owner controller is actively controlling the I/O module.

There is a noted difference in controlling input modules versus controlling output modules.

Controlling:	This ownership:	Description:
input modules	owner	An input module is configured by a controller that establishes a connection as an owner. This configuring controller is the first controller to establish an owner connection. Once an input module has been configured (and owned by a controller), other controllers can establish owner connections to that module. This allows additional owners to continue to receive multicast data if the original owner controller breaks its connection to the module. All other additional owners must have the identical configuration data and identical communications format that the original owner controller has, otherwise the connection attempt is rejected.
	listen-only	Once an input module has been configured (and owned by a controller), other controllers can establish a listen-only connection to that module. These controllers can receive multicast data while another controller owns the module. If all owner controllers break their connections to the input module, all controllers with listen-only connections no longer receive multicast data.
output modules	owner	An output module is configured by a controller that establishes a connection as an owner. Only one owner connection is allowed for an output module. If another controller attempts to establish an owner connection, the connection attempt is rejected.
	listen-only	Once an output module has been configured (and owned by one controller), other controllers can establish listen-only connections to that module. These controllers can receive multicast data while another controller owns the module. If the owner controller breaks its connection to the output module, all controllers with listen-only connections no longer receive multicast data.

Notes:

Communicating with Other Devices

Introduction

The MSG instruction asynchronously reads or writes a block of data to another device.

If the target device is a:	Select one of these message types:
Logix5000 controller	CIP Data Table Read
	CIP Data Table Write
I/O module that you configure using RSLogix 5000 software	Module Reconfigure
	CIP Generic
PLC-5 controller	PLC5 Typed Read
	PLC5 Typed Write
	PLC5 Word Range Read
	PLC5 Word Range Write
SLC controller MicroLogix controller	SLC Typed Read
	SLC Typed Write
Block-transfer module	Block-Transfer Read
	Block-Transfer Write
PLC-3 processor	PLC3 typed read
	PLC3 typed write
	PLC3 word range read
	PLC3 word range write
PLC-2 processor	PLC2 unprotected read
	PLC2 unprotected write

Caching Messages

Some types of messages use a connection to send or receive data. Some also give you the option of either leaving the connection open (cache) or closing the connection when the message is done transmitting. The following table shows which messages use a connection and whether or not you can cache the connection:

This type of message:	Using this communication method:	Uses a connection:	Which you can cache:
CIP data table read or write	CIP	X	X
PLC2, PLC3, PLC5, or SLC (all types)	CIP		
	CIP with Source ID		
	DH+	X	X
CIP generic	N/A	your option(1)	your option(1)
block-transfer read or write	N/A	X	X

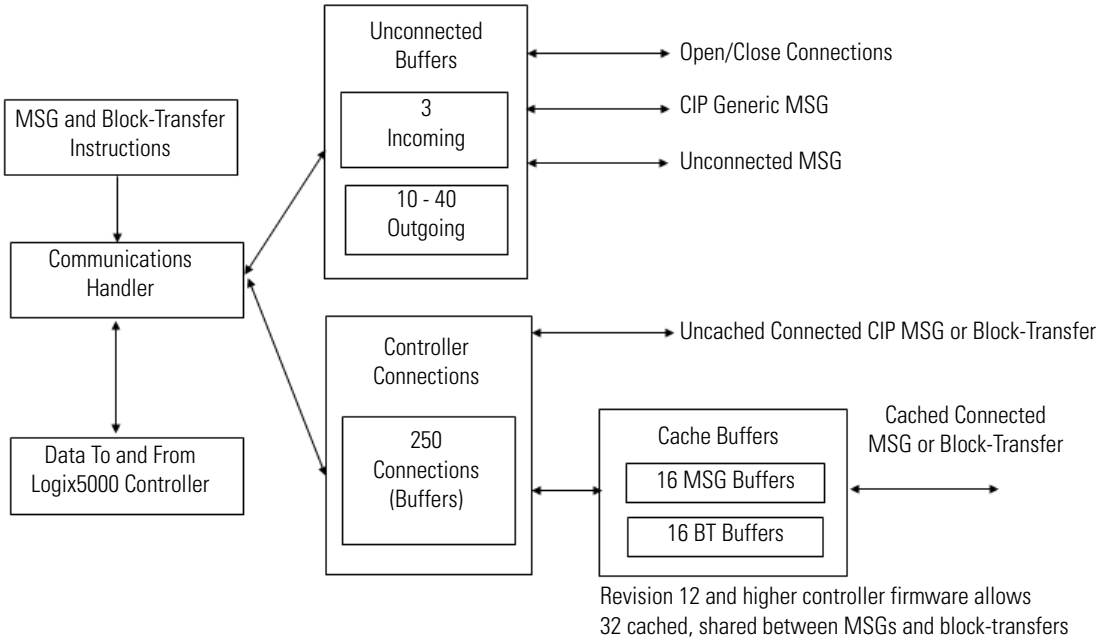
(1) You can connect CIP generic messages, but for most applications we recommend you leave CIP generic messages unconnected

A cached connection remains open until one of the following occurs:

- The controller goes to Program mode
- You rerun the message as uncached
- Another message is initiated and a cached buffer is needed
- An intermediate node in the connection goes down.

Message Buffers

A Logix5000 controller has buffers for unconnected messages and for cached messages. Buffers store incoming and outgoing message data until the controller can process the data.



Buffer:	Description:
10 outgoing unconnected buffers You can increase this to 40 by using a CIP Generic message instruction. See the MSG section in the *Logix 5000 Controllers General Instructions Reference Manual*, publication 1756-RM003	The outgoing unconnected buffers are for: • establishing I/O connections to local I/O modules and remote devices on ControlNet, EtherNet/IP, and Universal remote I/O networks • executing unconnected PLC2, PLC3, PLC5, or SLC (all types) messages over Ethernet or ControlNet (CIP and CIP with Source ID) • initiation of messaging over DH+ (uses 2 buffers, one to open the connection and one to transfer data) • initiation of uncached block transfers • initiation of uncached CIP read/write message instructions • initiation of cached block transfers • initiation of cached CIP read/write messages instructions • CIP Generic message instructions
3 incoming unconnected buffers	The incoming unconnected buffers are for: • initial receiving of a cached CIP message instruction • receiving an uncached CIP message instruction • receiving a message over DH+ • receiving a CIP Generic message instruction • receiving a read or write request from a ControlNet PanelView (unconnected messaging) • initial receiving of a read request from an Ethernet PanelView (connected messaging) • receiving a write request from an Ethernet PanelView (unconnected messaging) • receiving a initial request from RSLogix 5000 to go online • initial receiving of RSLinx connections
16 cached message buffers 16 cached block-transfer buffers	The cached buffers are outgoing buffers for cached messages and cached block-transfers. A cached connection helps message performance because the connection is left open and does not need to be reestablished next time it is executed. If you cache more than 16 messages in either set of buffers, the controller looks at the current buffers to determine how to deal with the additional cached messages. The controller will look for a connection that has been inactive for the longest time and close that connection and allow a new one to take its place. But if all 16 cached connections are in use, the message will use one of the 10 unconnected out going buffers. If all the unconnected buffers are in use, the message instruction will error with code 301 (No Buffer Memory) or 302 (Bandwidth Not Available). With revision 12 and higher controller firmware, you can cache 32 messages. For optimum performance, do not cache more than 32 messages. If you cache more than 32 messages, the controller looks for a connection that has been inactive for the longest time, closes that connection, and allows a new connection take its place. The controller will close a cached message or block-transfer, depending on which has been inactive the longest. If all 32 cached connections are in use, the message will use one of the unconnected out going buffers. The first time a cached message is executed, it uses one of the 10 out going unconnected buffers. When the connection is established it will then move into the appropriate cached buffer area.

In controllers with firmware revision 12 or earlier, the maximum number of messages you can have active at the same time depends on the type of message. If you are doing cached block transfers, you have memory set aside for 16 cached block transfer connections. If you are doing Logix-to-Logix or other message styles that can be cached, you have another set of 16 cached connections. You then still have the 10 unconnected out going buffers to use. So you could have 42 messages active at once.

Using outgoing unconnected buffers

Buffers:	Use:
1-10	The first 10 buffers (default) are shared for unconnected messaging, initiating connected messaging, establishing I/O connections, and establishing produced/consumed connections.
11	The 11th buffer is dedicated to establishing I/O and produced/consumed connections.
12-40	The 12th to the 40th buffers are used only for initiating connected messages and executing unconnected messages. To increase the outgoing buffers to a value higher than 11, execute a CIP generic message to configure that change.

Guidelines for Messages

Message tags must exist as controller-scoped, base tags	The information in a message tag is accessed by the operating system asynchronously to the program scan. In addition to the visible fields within the message tag, there are hidden attributes only referenced by the background operating system.
You can have more than 32 messages in a program	The controller supports 32 active messages at a time. If you determine that there are more than 32 messages, you will not be able to keep them cached. You will need extra programming to ensure that no more than 32 messages are active at the same time. Prior to controller revision 12, the controller supported 16 active messages at a time.
You can use a message to send a large amount of data	Even though there are network packet limitations (such as 500 bytes on ControlNet and 244 bytes on DH+), the controller can send a large amount of data from a single MSG instruction. When configuring the message, select an array as the source/destination tags and select the number of elements (as many as 32,767 elements) you want send. The controller automatically breaks the array into small fragments and sends all the fragments to the destination. On the receiving side, the data appears in fragments, so some application code may be required to detect the arrival of the last piece.

Guidelines for Managing Message Connections

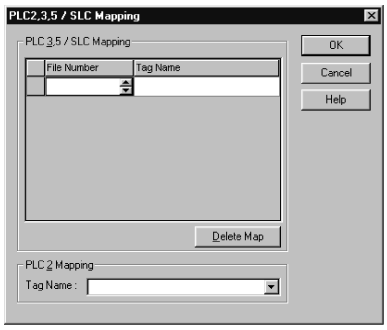
Create user-defined structures or arrays	User-defined structures let you organize your data to match your machine or process. • One tag contains all the data related to a specific aspect of your system. This keeps related data together and easy to locate, regardless of its data type. • Each individual piece of data (member) gets a descriptive name. This automatically creates an initial level of documentation for your logic. • You can use the structure to create multiple tags with the same data lay-out. • RSLinx optimizes user-defined structures more than stand-alone tags.
Cache connections when appropriate	If a message executes repeatedly, cache the connection. This keeps the connection open and optimizes execution time. Opening a connection each time the message executes increases execution time. If a message executes infrequently, do not cache the connection. This closes the connection upon completion of the message, which frees up that connection for other uses.
Use one message instruction multiple times for multiple devices	Each message uses one connection, regardless of how many devices are in the message path. To conserve connections, you can configure one message instruction to sequentially read from or write to a different device each time it executes. On each execution, the instruction breaks its connection from one device and re-establishes the connection to a subsequent device.

The system overhead timeslice percentage you configure for the controller determines the percentage of controller time (excluding the time for periodic and event tasks) that is devoted to communication and background functions. This includes sending and receiving messages. For information on specifying a system overhead percentage, see "*Selecting a System Overhead Percentage*" on page 2-10.

Guidelines for Block-Transfer Messages

Distribute 1771 analog modules across multiple chassis	Distributing 1771 analog modules across multiple chassis reduces the number of block-transfers that a single 1771-ACN or 1771-ASB module needs to manage.
Isolate different 1771 chassis on different networks	Isolating different chassis onto different networks diversifies the communications so that no single network or communication module has to deal with all of the communications.
Increase ControlNet unscheduled bandwidth	If communicating over ControlNet, increase the amount of ControlNet unscheduled bandwidth to permit additional time on the network for data exchange.
Increase the system overhead timeslice percentage	Increase the Logix5000 controller's system overhead timeslice to allocate more CPU time to communication processing from the continuous task.

Mapping Tags



A Logix5000 controller stores tag names on the controller so that other devices can read or write data without having to know physical memory locations. Many products only understand PLC/SLC data tables, so the Logix5000 controller offers a PLC/SLC mapping function that lets you map Logix tag names to memory locations.

- You only have to map the file numbers that are used in messages; the other file numbers do not need to be mapped.
- The mapping table is loaded into the controller and is used whenever a "logical" address accesses data.
- You can only access controller-scoped tags (global data).

When mapping tags:

- Do not use file numbers 0, 1, and 2. These files are reserved for Output, Input, and Status files in a PLC-5 processor.
- Use PLC-5 mapping only for tag arrays of data type INT, DINT, or REAL. Attempting to map elements of system structures may produce undesirable effects.
- Use these file types and identifiers:

For this Logix5000 array type:	Use this PLC file identifier:
INT array	N or B
DINT array	L
REAL array	F

Optimizing an Application for Motion Control

Introduction

The Logix5000 controller contains a high-speed motion task which executes motion commands (relay ladder and structured text) and generates position and velocity profile information. The controller sends this profile information to one or more motion modules. RSLogix 5000 programming software provides complete axis configuration and motion programming support.

For more information on motion, see:

- The Motion Book
- *Logix5000 Controllers Motion Instructions Reference Manual*, publication 1756-RM007
- *ControlLogix Motion Module Setup and Configuration Manual*, 1756-UM006

Coarse Update Rate

The coarse update rate determines the periodic rate at which the motion task executes to compute the servo commanded position, velocity, and accelerations to be sent to the motion modules when executing motion instructions.

To calculate the coarse update rate:

- 2 * (task execution time + number of actions for every axis)
- divide the result by 1000 and round to the nearest msec

If the coarse rate is too small, the controller might not have time to execute non-motion logic. As a general rule, one millisecond per axis is required by the motion task in order to allow the controller reasonable execution time.

The motion planner takes almost its entire minimum coarse iteration time. The coarse iteration time is minimally set 1 msec per axis. So if you have a periodic task running every 5 msec and 2 axes of motion, the motion planner runs twice consuming close to 4 of the 5 msec. In this case, it's possible to never finish executing the periodic task.

Axis Limits

Controller:	Supported Motion Modules and Axes:	Applications:
ControlLogix	1756-M03SE (3 axes)	RA SERCOS drives
	1756-L60M03SE (3 axes) 1756-L60 controller with embedded SERCOS interface	RA SERCOS drives
	1756-M08SE (8 axes)	RA SERCOS drives
	1756-M16SE (16 axes)	RA SERCOS drives
	1756-M02AE (2 axes)	RA and third party: • analog command signal • quadrature encoder feedback
	1756-HYD02	RA and third party: • analog command signal • linear transducer feedback
	1756-M02AS	RA and third party: • analog command signal • SSI feedback
SoftLogix	1784-PM16SE (16 axes) • maximum of four 1784-PM16SE cards per computer • associate only one 1784-PM16SE card with one controller	RA SERCOS drives
	1784-PM02AE (2 axes) • maximum of four 1784-PM02AE cards per computer • maximum of four 1784-PM02AE cards can be associated with one controller • cannot associate a 1784-PM02AE motion card with the same controller as a 1784-PM16SE card	RA and third party: • analog command signal • quadrature encoder feedback

Performance Limits

The motion planner interrupts all other tasks, regardless of priority.

- The number of axes and coarse update period for the motion group effect how long and how often the motion planner executes.
- If the motion planner is executing when a task is triggered, the task waits until the motion planner is done.
- If the coarse update rate occurs while a task is executing, the task pauses to let the motion planner execute.

Motion Event Task Triggers

An event task executes automatically based on a preconfigured event occurring. There are different motion-based events.

To trigger an event task when:	Use this trigger:	With these considerations:
registration input for an axis turns on (or off)	Axis Registration 1 or 2	• In order for the registration input to trigger the event task, first execute a Motion Arm Registration (MAR) instruction. This lets the axis detect the registration input and in turn trigger the event task. • Once the registration input triggers the event task, execute the MAR instruction again to re-arm the axis for the next registration input. • If the scan time of your normal logic is *not* fast enough to re-arm the axis for the next registration input, consider placing the MAR instruction within the event task.
axis reaches the position that is defined as the watch point	Axis Watch	• In order for the registration input to trigger the event task, first execute a Motion Arm Watch (MAW) instruction. This lets the axis detect the watch position and in turn trigger the event task. • Once the watch position triggers the event task, execute the MAW instruction again to re-arm the axis for the next watch position. • If the scan time of your normal logic is *not* fast enough to re-arm the axis for the next watch position, consider placing the MAW instruction within the event task
motion planner completes its execution	Motion Group Execution	• The coarse update period for the motion group triggers the execution of both the motion planner and the event task. • Because the motion planner interrupts all other tasks, it executes first. If you assign the event task as the highest priority task, it executes after the motion planner.

For information on configuring an event task, see Chapter 2 "*Dividing Logic into Tasks, Programs, and Routines*."

Notes:

Optimizing an Application for Use with HMI

Introduction

Rockwell Automation offers several HMI (human-machine interface) platforms:

Platform:	Description:
PanelView Plus	dedicated, machine-level HMI
RSView ME	open, machine-level HMI
RSView SE Station	single-workstation, supervisory-level HMI
RSView SE Distributed	multi-server, multi-client, supervisory-level HMI
RSView32	single-station or single-server, multi-client, supervisory-level HMI
RSLinx software	software products that provide plant-floor device connectivity for HMI applications includes: • RSLinx Classic, also known as RSLinx 2.x • RSLinx Enterprise

Deciding how to implement HMI

Method:	Benefits:	Considerations:
Single HMI	• all HMI/EOI support this method • limited number of controller connections • no server to setup and manage	• single point of failure for visualization • only one person can monitor a single display at a time
Multiple, Independent HMI	• all HMI/EOI support this method • the same HMI screens can be viewed at multiple stations • multiple people can monitor different parts of system simultaneously • each HMI gets its own data • no server to setup and manage	• more controller connections are required • additional burden on controller to service all communications (program scan impact) • no sharing of data • adding additional HMIs has larger increase on system
Client/Server HMI:	• the same HMI screens can be viewed at multiple stations • server collects data for multiple clients • fewer controller connections required • impact on system is smaller than with multiple HMIs	• additional server computer to administer • server is a single point of failure for all HMIs • little communications overhead savings if each client wants different data • adds communications

Most third-party HMIs are limited to direct communications similar to the multiple HMI method above.

Guidelines for HMI Applications

Limit the number of servers on one machine	On one machine, the maximum is: • one HMI server • one data server
Maximum of two HMI servers per application	Configure no more than two HMI servers per application: • use one computer for each server • maximum of 20,000 tags per HMI server
Maximum of two data servers per application	Configure no more than two data servers per application: • use one computer for each server
Maximum of 20 HMI clients per application	Configure no more than 20 HMI clients per application: • RSLinx Classic supports a maximum of 10 HMI clients • RSLinx Enterprise supports a maximum of 20 HMI clients See the "*Comparison of RSLinx Classic and RSLinx Enterprise*" on page 9-5.

Comparison of RSView32 and RSView Enterprise

HMI product:	Benefits:	Considerations:
RSView32	• support Windows NT, Windows 2000, and Windows XP	• RSView32 only supports development for PC-based HMIs • must use PanelBuilder software for PanelView terminals
RSView Enterprise	• supports Windows 2000 and Windows XP • single RSView Studio development environment for PC-based HMIs, PanelView Plus terminals, and VersaView CE terminals • FactoryTalk enabled	• does not support Windows NT

How RSLinx Software Communicates with Logix5000 Controllers

Important: Unless otherwise indicated, references to RSLinx software include both RSLinx Classic software and RSLinx Enterprise software

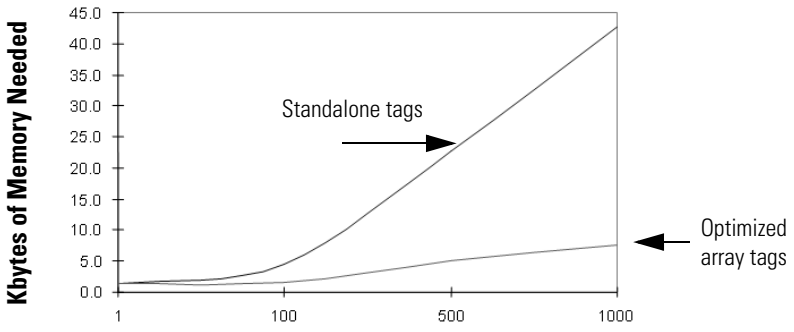
RSLinx software acts as a data server to optimize communications to HMI applications. RSLinx software groups data items into a single network packet to reduce the number of messages that get sent over the network and that need to be processed by a controller.

1. When RSLinx software first connects to a Logix5000 controller, it queries the tag database and uploads definitions for all controller-scoped tags. If there are multi-layer, user-defined structures that are controller-scoped, RSLinx software just queries the upper layer.

2. When the HMI client requests data, RSLinx software queries the definitions for program-scoped tags and the lower layers of multi-layer user-defined structures.

3. RSLinx software receives requests for data items from local or remote HMI/EOI clients and combines multiple requests in optimized packets. Each data item is a simple Logix tag, array or user-defined structure. Each optimized packet can be as large as 480 bytes of data and can contain one or more data items.

4. The Logix5000 controller allocates unused system RAM to create an optimization buffer to contain the requested data items.
   - a single optimization buffer can contain as much data as will fit into a single 480-byte packet (optimization is limited to 480 bytes)
   - currently, RSLinx Enterprise only provides optimization for array tags
   - if you use RSLogix 5000 software to monitor controller RAM, you can see used memory increase
   - the controller creates an optimization buffer for each RSLinx optimization packet in the scan.



Guidelines for RSLinx Software

Use RSLinx software as the data server for multiple HMIs	For multiple HMI stations: • leverage remote OPC (RSLinx Classic software) or FactoryTalk (RSLinx Enterprise software) for data collection • only the RSLinx data server should have an active topic • do not configure or use topics on the HMI stations • RSLinx software does not need to be on the HMI stations
Do not use too many RSLinx stations	The performance of tag collection decreases as the more RSLinx stations collect data from the same controller. Use an RSLinx Gateway station and have the other data collection stations use remote OPC for data collection.
Account for delay time when adding/removing scanned tags	When switching from one HMI screen to another, it takes time to put items in the controller on scan and take items off scan. Part of this time delay is due to the controller allocating system RAM for the optimization buffer. To eliminate this delay, when switching between HMI screens, put the items in the HMI screens on scan and leave them on scan. For example, you can create a data log to keep the items on scan. Then when switching between HMI screens, data collection continues without interruption. RSLinx Enterprise and RSView SE software account for this time delay. When HMI screens change, these applications deactivate tags rather than remove them from scan.

Comparison of RSLinx Classic and RSLinx Enterprise

Comparison:	RSLinx Classic (RSLinx 2.x) Software:	RSLinx Enterprise Software:
Supported platforms	• Windows 98 • Windows ME • Windows NT • Windows 2000 • Windows XP	• Windows CE • Windows 2000 • Windows XP
Architecture	single-threaded	multi-threaded
Data server	OPC data server preferred data server for PLC/SLC platforms and applications requiring complex network routings maximum 10 clients per data server	Factory Talk Live data server preferred data server for Logix5000 platforms maximum 20 clients per data server
PLC/SLC systems	maximum 20 controllers per data server via Ethernet	maximum 20 controllers per data server via Ethernet
Logix5000 systems	maximum: • 10 controllers per data server via Ethernet • 10,000 active (on-scan) tags per data server • 3 RSLinx data servers per controller	maximum: • 20 controllers per data server via Ethernet • 20,000 active (on-scan) tags per data server • 3 RSLinx Enterprise data servers per controller
User interface and event logs	yes	currently no
Benefits	• supports topic switching with redundant ControlLogix system • support used-defined tag optimization • RSLinx Gateway consolidates multiple HMI requests to reduce network traffic • works with integrated OPC server	• uses 4 read and 1 write uni-directional connections (fewer than RSLinx software) • automatically handles Logix tag changes • FactoryTalk Live consolidates multiple HMI requests to reduce network traffic
Considerations	• requires HMI to be restarted if Logix5000 controller is reloaded with changes to tags on scan • uses 4 bi-directional connections	• does not support topic switching with redundant ControlLogix system • optimization limited to array tags • does not yet support OPC • ActiveX faceplates require a separate OPC server

Guidelines for Configuring Controller Tags

Use INT data types with third party products	Most third party operator interface products do not support DINT (32-bit) data types. However, there are additional performance and memory-use considerations when using INT data types. See "*Guidelines for Data Types*" on page 3-2. RSView supports native Logix5000 data types (including BOOL, SINT, INT, DINT, and REAL), structures, and arrays.
Group related data in arrays	Most third party operator interface products do not support user-defined structures. Arrays also ensure that data is in contiguous memory, which optimizes data transfer between the controller and RSLinx or other operator interface. Arrays of tags transfer more quickly and take up less memory than groups of individual tags.
Map tags to PLC addresses	To optimize data transfer between the controller and RSLinx or other operator interface, use PLC mapped tags. The RSLinx topic must have the Optimize Poke Packets enabled. The RSView application must write the values through a DownloadAll command or the WritePendingValues VBA method.
Use RSLinx OPC services	Use RSLinx OPC services to bundle multiple tag requests into a single message to reduce communications overhead. OPC provides better optimization than DDE.

Referencing controller data from RSView

This table shows how to reference data in RSView tag address.

Logix5000 Array Data Type:	Description:	PLC File Identifier:	RSView Tag Data Type:
INT	16-bit integer	N	Integer
DINT	32-bit integer	L	Long Integer
SINT	8-bit integer	A	Byte
REAL	floating point	F	Floating Point
BOOL	value of 0, 1, or -1	B	Digital

When addressing a Logix5000 string tag, use the address syntax "[OPC_Topic]StringTag.Data[0],SC82" to address a SINT array. The string data is stored in the SINT array ".Data" of the string tag, and you address the first element of this array (".Data[0]"). The maximum number of characters in a STRING tag is 82. If you need more characters than that, create your own user-defined structure to hold the characters. See "*Guidelines for String Data Types*" on page 3-10.

Optimizing an Application for Process Control

Introduction

The Logix5000 controller integrates a function block diagram editor and several process control instructions. The controller can generally execute more loops than typical applications require.

Comparison of PID and PIDE Instructions

The function block PIDE instruction offers additional enhancements over the relay ladder PID instruction:

Enhanced PID (PIDE):	Standard PID:
velocity form algorithm which works on change in error value This algorithm is the same type as used in most DCS systems. The algorithm also makes it easier to implement adaptive gains.	position form algorithm which works on error values
full set of modes: • program/operator control • cascade/ratio mode • auto mode • manual mode • override mode • hand mode	limited set of modes • auto mode • software manual mode (similar to PIDE manual mode) • manual mode (similar to PIDE hand mode)
available selection of timing modes: • periodic • oversample • real time sampling	no timing modes
handling for PV/CV faults The PIDE block has built-in PVFault and CVFault members.	no handling for PV/CV faults
full bumpless transfer into and out of cascade mode	no bumpless transfer into or out of cascade mode

Guidelines for Programming PID Loops

Place PID loops in a periodic task	Configure the periodic tasks to execute at the desired rate.
Estimate the number of loops based on task execution time	Estimate the number of PID loops that can be executed as: (execution time of periodic tasks in msec) / 2 This leaves sufficient time for the controller to manage other logic in lower-priority tasks.

Estimating number of loops

The number of loops depends on the execution time of the periodic task as well as the controller.

	Periodic Task Execution Times (msec)						
Controller	**10**	**20**	**40**	**100**	**250**	**500**	**1000**
1756-L55	6	13	26	64	161	322	644
1756-L6x	18	36	72	180	450	899	1799

Advanced Process Instructions

Instruction:	Description:
Alarm (ALM)	provides alarming for any analog signal.
Enhanced PID (PIDE)	provides enhanced capabilities over the standard PID instruction. The instruction uses the velocity form of the PID algorithm. The gain terms are applied to the change in the value of error or PV, not the value of error or PV.
Ramp/Soak (RMPS)	provides for a number of segments of alternating ramp and soak periods.
Scale (SCL)	converts an unscaled input value to a floating point value in engineering units.
Position Proportional (POSP)	opens or closes a device by pulsing open or close contacts at a user-defined cycle time with a pulse width proportional to the difference between the desired and actual positions.
Split Range Time Proportional (SRTP)	takes the 0-100% output of a PID loop and drives heating and cooling digital output contacts with a periodic pulse.
Lead Lag (LDLG)	provides a phase lead-lag compensation for an input signal.
Function Generate (FGEN)	converts an input based on a piece-wise linear function.
Totalizer (TOT)	provides a time-scaled accumulation of an analog input value.
Deadtime (DEDT)	performs a delay of a single input. You select the amount of deadtime delay.
Discrete 2-State Device (D2SD)	controls a discrete device which has only two possible states such as on/off, open/closed, etc.
Discrete 3-State Device (D3SD)	controls a discrete device having three possible states such as fast/slow/off, forward/stop/reverse, etc.

Faceplates

The RSLogix 5000 programming software includes faceplates for some function block instructions. These faceplates are Active-X controls that read the entire data structure for the associated instruction You can use these faceplates with RSView software or any other application that acts as an Active-X container.

IMPORTANT RSLogix 5000 programming software is not a valid Active-X container.

The faceplates communicate with the controller via the RSLinx OPC server. The RSLinx OPC server is not available in the RSLinx Lite software that comes with RSLogix 5000 programming software. You have to purchase a package such as RSLinx OEM, Professional, or Gateway.

These instructions have faceplates:

- Alarm (ALM)
- Enhanced Select (ESEL)
- Totalizer (TOT)
- Ramp/Soak (RMPS)
- Discrete 2-State Device (D2SD)
- Discrete 3-State Device (D3SD)
- Enhanced PID (PIDE)

A

Term:	Definition:
atomic data type	BOOL, SINT, INT, DINT, and REAL data types.

B

Term:	Definition:
buffer	A temporary memory area used for queuing incoming and outgoing messages. The buffer area of a device determines how many messages can be queued for processing.

C

Term:	Definition:
cache	To leave a connection open for a MSG instruction that executes repeatedly.
coarse update rate	Determines the periodic rate at which the motion task executes to compute the servo commanded position, velocity, and accelerations to be sent to the motion modules when executing motion instructions.
compound data type	array, structure, and string data types.
connection	A communication link between two devices, such as between a controller and an I/O module, PanelView terminal, or another controller: • connections are allocations of resources that provide more reliable communications between devices than unconnected messages • you indirectly determine the number of connections the controller uses by configuring the controller to communicate with other devices in the system
consumed tag	A tag that receives the data that is broadcast by a produced tag over an EtherNet/IP network, ControlNet network, or ControlLogix backplane. A consumed tag must be: • controller scope • same data type (including any array dimensions) as the remote tag (produced tag) See *produced tag*.
controller scope	Data accessible anywhere in the controller. The controller contains a collection of tags that can be referenced by the routines and alias tags in any program, as well as other aliases in the controller scope. See *program scope*.

D

Term:	Definition:
direct connection	A direct connection is a real-time, data transfer link between the controller and an I/O module. The controller maintains and monitors the connection with the I/O module. Any break in the connection, such as a module fault or the removal of a module while under power, sets fault bits in the data area associated with the module. See *rack-optimized connection*.

E

Term:	Definition:
element	An addressable unit of data that is a sub-unit of a larger unit of data. A single unit of an array or structure.
explicit	A connection that is non-time critical and is request/reply in nature. Executing a MSG instruction or executing a program upload are examples of explicit connections. Explicit refers to basic information (source address, data type, destination address, etc.) that is included in every message. See *implicit*.

I

Term:	Definition:
implicit	A connection that is time critical in nature. This includes I/O and produced/consumed tags. Implicit refers to information (source address, data type, destination address, etc.) which is implied in the message but not contained in the message. See *explicit*.
index	A reference used to specify an element within an array.

L

Term:	Definition:
local connection	A connection to a module in a local chassis, extended-local chassis, or any of the I/O banks configured for the controller. Communication occurs across the backplane or virtual backplane and does not require an additional communication module or adapter.

M

Term:	Definition:
member	An element of a structure that has its own data type and name. • Members can be structures as well, creating nested structure data types. • Each member within a structure can be a different data type.

N

Term:	Definition:
network update time (NUT)	The repetitive time interval in which data can be sent on a ControlNet network. The network update time ranges from 2ms-100ms.

P

Term:	Definition:
postscan	A function of the controller where the logic within a program is examined before disabling the program in order to reset instructions and data.
prescan	Prescan is an intermediate scan during the transition to Run mode. • The controller performs prescan when you change from Program mode to Run mode. • The prescan examines all programs and instructions and initializes data based on the results. • Some instructions execute differently during prescan than they do during the normal scan.
produced tag	A tag that a controller is making available for use by other controllers. Produced tags are always at controller scope. See *consumed tag*.
program scope	Data accessible only within the current program. Each program contains a collection of tags that can only be referenced by the routines and alias tags in that program. See *controller scope*.

R

Term:	Definition:
rack-optimized connection	For digital I/O modules, you can select rack-optimized communication. A rack-optimized connection consolidates connection usage between the controller and all the digital I/O modules in the chassis (or DIN rail). Rather than having individual, direct connections for each I/O module, there is one connection for the entire chassis (or DIN rail). See *direct connection*.
remote connection	A connection to a module in a remote chassis or DIN rail. Communication requires a communication module and/or adapter.
requested packet interval (RPI)	When communicating over a the network, this is the maximum amount of time between subsequent production of input data. • Typically, this interval is configured in microseconds. • The actual production of data is constrained to the largest multiple of the network update time that is smaller than the selected RPI.

S

Term:	Definition:
scheduled connection	A scheduled connection is unique to ControlNet communications. A scheduled connection lets you send and receive data repeatedly at a predetermined rate, which is the requested packet interval (RPI). For example, a connection to an I/O module is a scheduled connection because you repeatedly receive data from the module at a specified rate. Other scheduled connections include connections to: • communication devices • produced/consumed tags On a ControlNet network, you must use RSNetWorx for ControlNet to enable all scheduled connections and establish a network update time (NUT).
structure	Some data types are a structure. • A structure stores a group of data, each of which can be a different data type. • Within a structure, each individual data type is called a member. • Like tags, members have a name and data type. • You create your own user-defined structure, using any combination of individual tags and most other structures. • To copy data to a structure, use the COP instruction.
system overhead timeslice	Specifies the percentage of controller time (excluding the time for periodic tasks) that is devoted to communication and background functions (system overhead):

U

Term:	Definition:
unconnected message	An unconnected message is a message that does not require connection resources. An unconnected message is sent as a single request/response.

Notes:

Available Publications

Logix5000 Platform:	Publications:
Logix5000 Controllers	• *Logix5000 Controllers Quick Start*, 1756-QS001 • *Logix5000 Controllers System Reference Manual*, 1756-QS107 • *EtherNet/IP Modules in Logix5000 Control Systems User Manual*, ENET-UM001 • *ControlNet Modules in Logix5000 Control Systems User Manual*, CNET-UM001 • *DeviceNet Modules in Logix5000 Control Systems User Manual*, DNET-UM004 • *Logix5000 Controllers General Instructions Reference Manual*, 1756-RM003 • *Logix5000 Controllers Process Control and Drives Instructions Reference Manual*, 1756-RM006 • *Logix5000 Controllers Motion Instructions Reference Manual*, 1756-RM007 • *Logix5000 Common Procedures Programming Manual*, 1756-PM001 • *Logix5000 Controllers Import/Export Reference Manual*, 1756-RM084G • *Converting PLC-5 or SLC 500 Logic to Logix5000 Logic Reference Manual*, 1756-RM085
ControlLogix Controllers	• *ControlLogix Controllers Installation Instructions*, 1756-IN101 • *ControlLogix System User Manual*, 1756-UM001 • *ControlLogix Motion Module Setup and Configuration Manual*, 1756-UM006
CompactLogix Controllers	• *1769-L31 CompactLogix Controllers Installation Instructions*, 1769-IN069 • *1769-L32, -L35E CompactLogix Controllers Installation Instructions*, 1769-IN020 • *1769-L20, -L30 CompactLogix Controllers Installation Instructions*, 1769-IN047 • *1769-L31, -L32E, -L35E CompactLogix System User Manual*, 1769-UM011 • *1769-L20, -L30 CompactLogix System User Manual*, 1769-UM007
FlexLogix Controllers	• *FlexLogix Controllers Installation Instructions*, 1794-IN002 • *FlexLogix System User Manual*, 1794-UM001
SoftLogix Controllers	• *SoftLogix Controllers Installation Instructions*, 1789-IN001 • *SoftLogix System User Manual*, 1789-UM002

RSLogix 5000 programming software includes PDF files of these publications, in addition to online help and a tutorial.

Rockwell Automation Support

Rockwell Automation provides technical information on the web to assist you in using our products. At http://support.rockwellautomation.com, you can find technical manuals, a knowledge base of FAQs, technical and application notes, sample code and links to software service packs, and a MySupport feature that you can customize to make the best use of these tools.

For an additional level of technical phone support for installation, configuration and troubleshooting, we offer TechConnect Support programs. For more information, contact your local distributor or Rockwell Automation representative, or visit http://support.rockwellautomation.com.

Installation Assistance

If you experience a problem with a hardware module within the first 24 hours of installation, please review the information that's contained in this manual. You can also contact a special Customer Support number for initial help in getting your module up and running:

United States	1.440.646.3223 Monday – Friday, 8am – 5pm EST
Outside United States	Please contact your local Rockwell Automation representative for any technical support issues.

New Product Satisfaction Return

Rockwell tests all of our products to ensure that they are fully operational when shipped from the manufacturing facility. However, if your product is not functioning and needs to be returned:

United States	Contact your distributor. You must provide a Customer Support case number (see phone number above to obtain one) to your distributor in order to complete the return process.
Outside United States	Please contact your local Rockwell Automation representative for return procedure.

www.rockwellautomation.com

Corporate Headquarters
Rockwell Automation, 777 East Wisconsin Avenue, Suite 1400, Milwaukee, WI, 53202-5302 USA, Tel: (1) 414.212.5200, Fax: (1) 414.212.5201

Headquarters for Allen-Bradley Products, Rockwell Software Products and Global Manufacturing Solutions
Americas: Rockwell Automation, 1201 South Second Street, Milwaukee, WI 53204-2496 USA, Tel: (1) 414.382.2000, Fax: (1) 414.382.4444
Europe: Rockwell Automation SA/NV, Vorstlaan/Boulevard du Souverain 36-BP 3A/B, 1170 Brussels, Belgium, Tel: (32) 2 663 0600, Fax: (32) 2 663 0640
Asia Pacific: Rockwell Automation, 27/F Citicorp Centre, 18 Whitfield Road, Causeway Bay, Hong Kong, Tel: (852) 2887 4788, Fax: (852) 2508 1846

Headquarters for Dodge and Reliance Electric Products
Americas: Rockwell Automation, 6040 Ponders Court, Greenville, SC 29615-4617 USA, Tel: (1) 864.297.4800, Fax: (1) 864.281.2433
Europe: Rockwell Automation, Brühlstraße 22, D-74834 Elztal-Dallau, Germany, Tel: (49) 6261 9410, Fax: (49) 6261 17741
Asia Pacific: Rockwell Automation, 55 Newton Road, #11-01/02 Revenue House, Singapore 307987, Tel: (65) 351 6723, Fax: (65) 355 1733

Publication 1756-RM094A-EN-P - May 2004 PN 957867-08

Copyright © 2004 Rockwell Automation, Inc. All rights reserved. Printed in the U.S.A.